Міністерство освіти і науки України
Сумський державний університет

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Конспект лекцій

Суми
Видавництво СумДУ
2010

Міністерство освіти і науки України
Сумський державний університет

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Конспект лекцій
для студентів спеціальності
080402 – Інформаційні технології проектування
заочної форми навчання

Затверджено
на засіданні секції
інформаційних технологій
проектування кафедри
інформатики з дисципліни
«Інформаційні технології
проектування».
Протокол № 08 від 30.03.2010 р.

Суми
Видавництво СумДУ
2010

Організація баз даних та знань: конспект лекцій для студентів заочної форми навчання/ Укладач А.В. Неня.– Суми: Вид-во СумДУ, 2010.– 109 с.

Секція інформаційних основ проектування.
Кафедра інформатики

Зміст

1 Основи проектування реляційних баз даних

1.1 Інформаційні системи з базами даних

1.1.1 Інформація й дані

Перш ніж перейти до обговорення поняття інформаційної системи (ІС), спробуємо з'ясувати, що ж розуміється під словом "інформація". Відповісти на це питання і просто, і складно: слово "інформація" пов'язане із широким колом понять.

Змістовна сторона поняття "інформація" дуже багатогранна й не має чітких семантичних меж. Однак завжди можна сказати, що можна з нею робити. Саме відповідь на це запитання найчастіше й цікавить як системних аналітиків і розроблювачів ІС, так і користувачів інформації (її основних споживачів).

З погляду як користувачів, так і розроблювачів ІС в інформації є одна важлива властивість - вона є одиницею даних, яка підлягає обробці. Звичайно інформація надходить споживачеві саме у вигляді даних: таблиць, графіків, малюнків, фільмів, усних повідомлень, які фіксують у собі інформацію певної структури й типу. Таким чином, дані виступають як засіб подання інформації у певній, фіксованій формі, придатній для обробки, зберігання й передачі. Хоча дуже часто терміни "інформація" й "дані" виступають як синоніми, варто пам'ятати про цю їхню істотну відмінність. Саме в даних інформація знаходить інтерпретацію у конкретній ІС.

При згадуванні про "форму" подання інформації варто сказати ще про одну, "людську" властивість інформації - її сприйняття різними категоріями людей. Дані можуть бути згруповані спільно у документ.

Документ може мати або не мати певної внутрішньої структури. Дані можуть бути відображені на екрані дисплея комп'ютера. Документи можуть мати аудіо- або відеоформу. Розробляючи ІС, ніколи не слід забувати, для кого вони (системи) створюються і хто буде їх використовувати. Форма подання інформації в ІС визначає також і категорії користувачів. ІС створюються для конкретних груп користувачів, тобто вони, як правило, проблемно-орієнтовані.

Інформація є даними, яким надається деякий зміст (інтерпретація) у конкретній ситуації у рамках деякої системи понять. Інформація подається за допомогою кодування даних і здобувається шляхом їхнього декодування та інтерпретації.

У цьому визначенні фіксується три основних перетворення інформації й даних у процесі їхньої обробки в ІС: інформація – дані, дані – дані, дані – інформація.

На рис. 1.1 наведені дві сторони визначення поняття інформації: функціональна й представницька. Перша загалом визначає коло дій над інформацією, а друга – результат виконання цих дій.

Рисунок 1.1 – Зміст поняття "інформація"

1.1.2 Інформаційні системи

Основною метою створення ІС є задоволення інформаційних потреб користувачів шляхом надання необхідної їм інформації на основі збережених даних. Потреба в інформації як такій не вичерпує поняття інформаційних потреб. Звичайно в поняття інформаційних потреб включають певні вимоги до якості інформаційного обслуговування й поводження системи в цілому (продуктивність, актуальність і надійність даних, орієнтація на користувача та ін.).

Під *інформаційною системою* розуміється організаційна сукупність технічних засобів, технологічних процесів і кадрів, що реалізують функції збору, обробки, зберігання, пошуку, видачі й передачі інформації.

Необхідність підвищення продуктивності праці у сфері інформаційної діяльності приводить до того, що в якості зовнішніх засобів зберігання й швидкого доступу до інформації найчастіше використовуються засоби обчислювальної техніки (цифровий і аналоговий) на основі комп'ютерів. Сучасні ІС - складні комплекси апаратних і програмних засобів, технології й персоналу, які ще називають автоматизованими інформаційними системами. Структурно ІС містять у собі апаратне (hardware), програмне (software), комунікаційне (netware), проміжного шару (middleware), лінгвістичне й організаційно-технологічне забезпечення.

Апаратне забезпечення ІС містить у собі широкий набір засобів обчислювальної техніки, передачі даних, а також цілий ряд спеціальних технічних пристроїв (пристрою графічного відображення інформації, аудио- і відеопристрою, засобу мовного уведення й т.д.). Апаратне забезпечення є основою будь-якої ІС.

Комунікаційне (мережне) забезпечення містить у собі комплекс апаратних мережних комунікацій і

програмних засобів підтримки комунікацій в ІС. Воно має істотне значення при створенні розподілених ІС й ІС на основі Інтернету.

Програмне забезпечення ІС забезпечує реалізацію функцій введення даних, їх розміщення на носіях, модифікації даних, доступ до даних, підтримку функціонування устаткування. Програмне забезпечення можна розділити на системне (яке завершує процес вибору апаратно-програмного рішення або платформи) і користувацьке (яке застосовується для вирішення завдань задоволення потреб користувача у комп'ютерному середовищі).

Лінгвістичне забезпечення ІС призначене для вирішення завдань формалізації змісту повнотекстової і спеціальної інформації для створення пошукового образу даних (профілю). У класичному змісті звичайно воно включає процедури індексування текстів, їхню класифікацію і тематичну рубрикацію. Найчастіше ІС, що містять складно-структуровану інформацію, містять у собі тезауруси термінів і понять. Сюди можна віднести й створення процесорів спеціалізованих формальних мов кінцевих користувачів, наприклад, мов для маніпулювання бухгалтерською інформацією і т.д. Найчастіше працям з розроблення лінгвістичного забезпечення не надається належного значення. Подібні недогляди найчастіше призводять до несприйняття користувачами самої інформації. Це стосується в першу чергу вузько спеціалізованих ІС.

У міру зростання складності і масштабів ІС важливу роль починає відігравати *організаційно-технологічне забезпечення*, що з'єднує різнорідні компоненти (апаратури, програми й персонал) у єдину систему й забезпечує процедури її керування й функціонування. Недооцінка цієї складової ІС найчастіше призводить до

зриву строків впровадження системи й виведення її на виробничі потужності.

На рис. 1.2 наведені функції ІС через її основні структурні компоненти.

Рисунок 1.2 – Визначення інформаційної системи

1.1.3 Основні підходи до обробки інформації в автоматизованих ІС

Одним з головних питань розроблення програмного забезпечення ІС є питання про співвіднесення програм і даних, тому що вирішення цього питання, в остаточному підсумку, визначає вибір алгоритмів обробки інформації, апаратних засобів і технологічної платформи. Фундаментальним принципом у вирішенні питання про співвіднесення програм і даних є концепція незалежності прикладних програм від даних, і неважливо, яка обробка даних передбачається: централізована або розподілена. Суть цієї концепції полягає не стільки у відділенні програм від даних, скільки у розгляді їх як самостійних взаємодіючих об'єктів.

Однією з останніх модифікацій цього принципу є концепція незалежності прикладних програм від даних разом із процедурами їхньої обробки (об’єктно-орієнтований підхід у програмуванні), що дозволяє вирішити ряд питань обробки даних, пов'язаних з інтерпретацією семантичного змісту даних.

Формування концепції БД і створення на її основі методу баз даних для вирішення завдань обробки інформації відбулося у 1962 році. До середини 60-х років минулого століття основною концепцією побудови програмного забезпечення були концепція файлової системи і так званий позадачний метод. Наприкінці 80-х років минулого століття була запропонована концепція об’єктно-орієнтованих баз даних й об’єктно-орієнтований підхід розроблення програм на основі обробки подій. На рис. 1.3 наведені основні ознаки для кожної з зазначених вище концепцій. На рис. 1.4 проведене зіставлення основних методів обробки даних.

Основний зміст позадачного методу зводиться до декомпозиції програми зі своїми окремими блоками даних та алгоритмами; методу баз даних – до наявних окремих описів логічної структури даних та єдиної точки зору щодо процедури обробки даних; об'єктно- орієнтованого методу – полягає в тому, що програми розглядаються як сукупність об'єктів, між якими відбувається обмін інформацією.

Об'єкту притаманні такі властивості:

- інкапсуляція – об'єкти наділяються структурою й мають певне поводження (набором операцій). Операції над об'єктами становлять його методи. Структура об'єкта захована від користувача, що маніпулює об'єктом через його операції. Об'єкт розглядається як абстракція реального світу Для того щоб об'єкт виконав деяку дію,

Рисунок 1.3 – Основні концепції обробки інформації

Рисунок 1.4 – Основні проблеми методів обробки інформації

йому потрібно надіслати повідомлення. Об'єкт взаємодіє з іншими об'єктами через події;

- спадкування – являє собою механізм, що дозволяє робити одні об'єкти з інших, при цьому властивості батьківського об'єкта зберігаються у нащадка;
- поліморфізм – різні об'єкти можуть одержувати однакові повідомлення, але реагувати на них по-різному відповідно до реалізації своїх однойменних методів.

1.1.4 Бази даних і системи керування базами даних

Базу даних у загальному випадку можна визначити як уніфіковану сукупність збережених і відтворених даних, що використовуються у рамках організації (Engles R.A., 1972 р.). Однак поняття БД не ґрунтується в цей час на єдиній концепції, скоріше це ціле сімейство пов'язаних між собою понять з ПО, програмного й апаратного забезпечення, аналізу й моделювання даних і додатків. Існує кілька визначень БД.

База даних (за Дж. Мартіном) є сукупністю взаємозалежних даних, які спільно використовуються декількома додатками й зберігаються з мінімальною регульованою надлишковістю. Дані запам'ятовуються таким чином, щоб вони у міру можливості не залежали від програм. Для обробки даних застосовується загальний керуючий метод доступу. Якщо БД не перетинаються за структурою, то говорять про систему баз даних.

База даних (відповідно до матеріалів комітету КОДАСІЛ) складається зі всіх екземплярів записів, екземплярів наборів записів і областей, які контролюються конкретною схемою. Під схемою можна розуміти карту всієї логічної структури БД.

Для розроблювача ІС істотним моментом при використанні концепції баз даних (БД) є та обставина, що

дані стають певним чином організовані, здобувають якусь упорядкованість і внутрішню структуру, а також те, що є деякий набір уніфікованих операцій обробки даних і декларативних засобів подання даних. До таких операцій варто віднести операції "Вставити" (Insert), "Додати" (Add), "Видалити" (Delete) і ряд інших. До декларативних засобів подання даних варто віднести мови визначення даних. Тобто використання даної концепції при створенні ІС припускає наявність мови визначення даних і мови маніпулювання даними, а також правил побудови інтерфейсів програм (додатків)із БД і користувачем.

Такий розподіл засобів маніпулювання даними і їхнього подання є деякою мірою умовним. Мова визначення даних служить для опису логічної структури (схеми) БД, а в деяких випадках і способів зберігання й доступу до даних. Мова маніпулювання даними надає алгоритмічні засоби побудови додатків для обробки елементів даних, які зберігаються в БД.

У разі застосування концепції БД для створення ІС природно виникає питання, - а хто або що повинне все це підтримувати? Таким чином, постає питання про Систему керування базою даних (СКБД). СКБД є складними програмними системами, що працюють на різних операційних платформах. Саме СКБД повинна надати засоби визначення й маніпулювання даними, зробивши дані незалежними від прикладних програм, що їх використовують.

До основних функцій СКБД слід віднести:

- забезпечення мовних засобів опису та маніпуляції даними;
- забезпечення підтримки логічної моделі даних;
- забезпечення взаємодії логічної та фізичної структур даних;
- забезпечення захисту та цілісності даних;

- забезпечення підтримки БД в актуальному стані.

Системою керування базами даних (Data-base Management System) називається сукупність програмних засобів, необхідних для використання БД і подання розробникам і користувачам безлічі різних подань даних.

1.1.5 Поняття про моделі даних

Подання інформації за допомогою даних вимагає уніфікованого підходу до поняття даних як незалежного об'єкта моделювання. Тому для розробника ІС вибір відповідної моделі даних є однією з найважливіших проблем. Вибір моделі даних спричиняє вибір засобів аналізу ПО як області реального світу, що підлягає вивченню й обробці. Модель даних обмежує можливість вибору СКБД, тому що, як правило, окремо взята модель підтримує певну модель даних. Таким чином, поняття моделі даних є одним з фундаментальних понять інформатики, від якого багато в чому залежать механізми реалізації ІС як програмно-апаратного комплексу.

Модель даних (Data Model) є логічною структурою даних, що становить притаманні цим даним властивості, незалежні від апаратного й програмного забезпечення й не пов'язані з функціонуванням комп'ютера.

Можна розглянути кілька аспектів моделювання в обробці даних:

- інформаційне моделювання:
- концептуальне моделювання (моделювання семантики ПО);
- логічне моделювання даних;
- фізичне моделювання:
- створення моделей доступу до даних;

• оптимізація фізичної організації даних в апаратному середовищі.

На рис. 1.5 ілюструється загальний зміст поняття моделі даних на цей час.

Рисунок 1.5 – Подання про інформаційну модель даних

Основні типи моделей й їхня еквівалентність

Наявність у СКБД певної структури даних приводить до поняття баз структурованих даних, тобто дані в таких БД повинні бути представлені як сукупність взаємозалежних елементів. Варто мати на увазі, що для кожного типу БД використовуються відповідні моделі даних.

У цей час для баз структурованих даних розрізняють три основні типи логічних моделей даних залежно від характеру підтримуваних ними зв'язків між елементами даних - мережну, ієрархічну й реляційну. Ознаками класифікації у цих моделях є: ступінь твердості (фіксації) зв'язку, математичне подання структури моделі і припустимих типів даних (див. таблицю 1.1).

Рис. 1.6 ілюструє особливості кожної моделі даних. При зіставленні моделей варто пам'ятати, що всі вони теоретично еквівалентні. Еквівалентність моделей полягає в тому, що вони можуть бути зведені одна до одної шляхом формальних перетворень.

Таблиця 1.1 – Загальні характеристики моделей даних

Модель даних	Характер зв'язків між об'єктами	Формальне подання
Мережна	Напівтверді зв'язки	Довільний граф
Ієрархічна	Тверді зв'язки	Деревоподібна структура
Реляційна	Мінливі зв'язки	Плоский файл

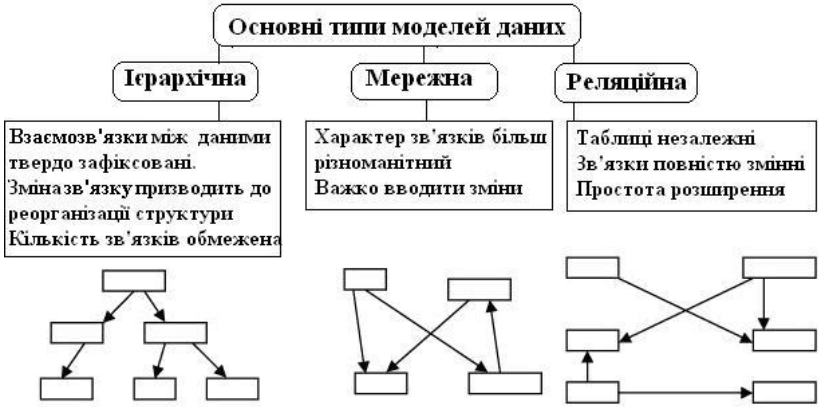


Рисунок 1.6 – Основні типи моделей даних

1.2 Предметна область БД і її моделі

1.2.1 Поняття ПО

Основним призначенням ІС є оперативне забезпечення користувача інформацією про зовнішній світ шляхом реалізації питально-відповідного відношення. Питально-відповідні відношення дозволяють виділити для

ІС певний її фрагмент - предметну область, - який буде втілений в автоматизованій ІС. Інформація про зовнішній світ подається в ІС у формі даних, що обмежує можливості змістовної інтерпретації інформації й конкретизує семантику її подання в ІС. Сукупність цих виділених для ІС даних, зв'язків між ними й операцій над ними утворить інформаційну й функціональну моделі ПО, що описують її стан з певною точністю. Інформаційна й функціональна моделі ПО є вхідними даними для процесу проектування БД.

Сукупність реалій (об'єктів) зовнішнього світу - об'єктів, про які можна питати, - утворює об'єктне *ядро ПО*, яке має онтологічний статус. Не можна одержати в ІС відповідь на запитання про те, що їй невідомо. Термін "об'єкт" є первинним поняттям. Синонімами терміна "об'єкт" є "реалія, сутність, річ". *Сутність ПО* є результатом абстрагування реального об'єкта шляхом виділення й фіксації набору його властивостей.

Прикладами сутностей (з погляду ІС) або об'єктів (з погляду зовнішнього світу) є окремий студент, група студентів, аудиторія, час занять, слова, числа, символи. Звичайно вважається, що бути об'єктом - означає бути дискретним і помітним.

З об'єктами пов'язано дві проблеми: ідентифікація й адекватний опис. Для ідентифікації використовують ім'я. Використовується тільки вказівна функція імені. *Ім'я*_- це прямий спосіб ідентифікації об'єкта. До непрямих способів ідентифікації об'єкта відносять визначення об'єкта через його властивості (характеристики або ознаки).

Об'єкти взаємодіють між собою через свої властивості, що породжує ситуації. *Ситуації* - це взаємозв’язки, які виражають взаємини між об'єктами. Ситуації у предметній області (ПО) описуються за

допомогою висловлювань про ПО з використанням виразів і обчисленнями предикатів, тобто формальної, математичної логіки.

Методи математичної логіки дозволяють формалізувати ці твердження і подати їх у вигляді, придатному для аналізу.

Приклад. Розглянемо висловлювання: студент Іванов А.А, народився у 1982 році. Воно виражає такі властивості об'єкта "Іванов А.А.":

у явному виді - рік народження;

у неявному - приналежність до студентів.

Перша властивість встановлює зв'язок між об'єктами "Іванов А.А." і "Рік народження", а друга - між об'єктами "Іванов А.А." і "Безліч студентів". Формалізація цього висловлювання представляється як результат присвоювання значень змінним, які входять у предикати:

НАРОДИВСЯ (Іванов А.А., 1982)

Є СТУДЕНТОМ (Іванов А.А.)

На рис. 1.7 наведений один із підходів до класифікації ситуацій у рамках ПО.

Рисунок 1.7 – Класифікація ситуацій ПО

Розрізняють статичні й динамічні ситуації. Прикладами статичних ситуацій є такі ситуації, як кольори, вік. Прикладами динамічних ситуацій є такі ситуації, як випекти хліб.

Наведена класифікація вводить у ПО два важливі аспекти - простір і час, до того ж час і як момент, і як інтервал. ПО існує у просторі і часі, тобто їй притаманні часові і просторові відношення і зв’язки. Слід розрізняти реальний час зовнішнього світу та його відображення у БД та у джерелах інформації. У БД взаємозв’язки, залежні від часу, фіксуються тільки після реєстрації в БД. Таким чином, ПО у кожний певний момент часу являє собою відокремлену сукупність визначених об’єктів і ситуацій, яку називають станом ПО.

Предметна область - це цілеспрямована первинна трансформація картини зовнішнього світу у деяку картину, певна частина якої фіксується в ІС як алгоритмічна модель фрагменту дійсності.

1.2.2 Інформаційна модель ПО БД

Інформаційна модель даних призначена для подання семантики ПО у термінах суб'єктивних засобів опису - сутностей, атрибутів, ідентифікаторів сутностей, супертипів, підтипів і т.д.

Інформаційна модель ПО БД містить такі основні конструкції:

- діаграми "сутність-зв'язок" (Entity - Relationship Diagrams);
- визначення сутностей;
- унікальні ідентифікатори сутностей;
- визначення атрибутів сутностей;
- відношення між сутностями;
- супертипи й підтипи.

Елементи інформаційної моделі даних ПО є вхідними даними для вирішення завдання проектування БД - створення логічної моделі даних.

Предметом інформаційної моделі є абстрагування об'єктів або явищ реального світу у рамках ПО, у результаті якого виявляються сутності (entity) ПО. Як правило, вони позначаються іменником природної мови.

Сутність описується за допомогою даних, іменованих властивостями або атрибутами (attributes) сутності. Як правило, *атрибути* є визначеннями у висловленні про сутності й позначаються іменниками природної мови. Сутності вступають у зв'язки один з одним через свої атрибути. Кожна група атрибутів, що описують один реальний прояв сутності, являє собою екземпляр (instance) сутності. Іншими словами, екземпляри сутності - це реалізації сутності, що відрізняються одне від одного й допускають однозначну ідентифікацію.

Одним з основних комп'ютерних засобів розпізнавання сутностей у базі даних є присвоєння сутностям ідентифікаторів (Entity identifier). Часто ідентифікатор сутності називають ключем. Завдання вибору ідентифікатора сутності є суб'єктивним завданням. Оскільки сутність визначається набором своїх атрибутів, то для кожної сутності доцільно виділити таку підмножину атрибутів, що однозначно ідентифікує дану сутність.

Завдання розробника БД - забезпечити при збереженні екземплярів сутності у БД наявність у кожного її нового екземпляра унікального ідентифікатора. *Унікальний ідентифікатор сутності* - це атрибут сутності, що дозволяє відрізняти одну сутність від іншої. Якщо сутність має кілька унікальних ідентифікаторів, так званих можливих ключів, то розробник повинен обрати первинний ключ сутності.

Розрізняють однозначні й багатозначні атрибути. Однозначними є атрибути, які в межах конкретного екземпляра сутності мають тільки одне значення. У протилежному разі вони вважаються багатозначними.

Кожен атрибут сутності має домен (domain). *Домен* - це вираз, який визначає значення, дозволені для даного атрибута. Іншими словами, домен - це область значень атрибута. Розробник БД повинен проконтролювати, щоб в інформаційній моделі ПО для кожного атрибута сутностей був визначений домен.

Сутності не існують окремо одна від одної. Між ними є реальні відношення (Relationship), і вони повинні бути відбиті в інформаційній моделі ПО. При виділенні відношень акцент робиться на фіксацію зв'язків і їх характеристик. *Відношення* (зв'язок) являє собою з'єднання (взаємовідношення) між двома або більше сутностями. Кожен зв'язок реалізується через значення атрибутів сутностей. Звичайно зв'язок позначається дієсловом. Кожен зв'язок також повинен мати свій унікальний ідентифікатор зв'язку.

Розробник БД повинен проконтролювати, щоб зв'язок між сутностями здійснювався через точно зазначені атрибути, які будуть визначати унікальний ключ зв'язку. Вибір ключів сутностей - одне з найважливіших проектних рішень, що має бути зробленим розробником при переході від інформаційної моделі ПО до логічної моделі БД.

Зв'язки характеризуються ступенем зв'язку і класом належності сутності до зв'язку. Ступінь (потужність) зв'язку - це відношення числа сутностей, що беруть участь в утворенні зв'язку. Існують такі типи: "один-до-одного", "один-до-множини", "множина-до-множини".

Типовою формою документування інформаційної моделі ПО є діаграми "сутність-зв'язок" (ER-діаграми). ER-діаграма дозволяє графічно представити всі елементи

інформаційної моделі згідно із простим, інтуїтивно зрозумілим, але чітко визначеним правилом - нотаціям. Далі ми будемо користуватися умовними позначеннями, прийнятими в методології інформаційного проектування.

Сутність на ER-діаграмі наводиться прямокутником з ім'ям у верхній частині. Будемо використовувати англійські слова для іменування елементів моделі.

Рисунок 1.8 – Подання сутності Person (персонал) на ER-діаграмі з атрибутами й унікальним ідентифікатором сутності

У прямокутнику перераховуються атрибути сутності, при цьому атрибути, що становлять унікальний ідентифікатор сутності, підкреслюються.

Домени призначаються аналітиками й фіксуються в спеціальному документі - словнику даних (Data Dictionary). На стадіях розроблення логічної й фізичної моделей реляційної БД домени уточнюються у сутностях на ER-діаграмі.

Розробник БД повинен ретельно вивчити домени кожного атрибута з погляду на можливість їх реалізації у СКБД.

PERSON	
PERSON ID	INTEGER
F_NAME	CHAR(30)
L_NAME	CHAR(20)
BIRTHDAY	DATE
HEIGHT	HUMBER(5,2)
WEIGHT	HUMBER(4,1)
SEX	HUMBER(1)
SSN	CHAR(30)
PHOTO	LONG RAW

Рисунок 1.9 – Візуалізація визначення доменів атрибутів на ER-діаграмі при створенні фізичної моделі реляційної БД

Відношення (зв'язок) сутностей на ER-діаграмі зображується лінією, що з'єднує ці сутності. Ступінь зв'язку зображується за допомогою символа "пташина лапка", що вказує на те, що у зв'язку бере участь багато (N) екземплярів сутності, і одинарною горизонтальною рискою, що вказує на те, що у зв'язку бере участь один екземпляр сутності.

Відношення читається вздовж лінії або ліворуч праворуч, або праворуч–ліворуч. На рис. 1.10 наведене таке відношення: кожна спеціальність із створення повинна бути зареєстрована за певною фізичною особою (персоною), фізична особа може мати одну або більше спеціальностей щодо створення.

1.2.3 Функціональна модель ПО БД

Другим ключовим моментом створення ІС з метою автоматизації інформаційних процесів організації є аналіз функціональної взаємодії об'єктів автоматизації. Аналітики наводять результати у вигляді функціональної моделі ПО БД. Склад функціональної моделі істотно залежить від

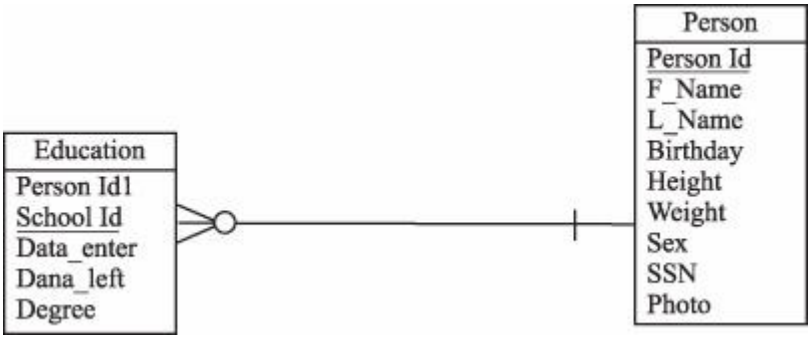


Рисунок 1.10 – Подання відношення між двома сутностями на ER-діаграмі

контексту конкретного IT-проекту і може бути представлений за допомогою досить широкого спектра документів у вигляді текстової і графічної інформації.

Функціональна модель призначена для опису процесів обробки даних у рамках виділеної ПО з різних точок зору.

Визначимо *функціональну модель* ПО БД як сукупність деяких моделей, призначених для опису процесів обробки інформації. Будемо називати ці моделі конструкціями функціональної моделі. Нижче наведений перелік основних конструкцій функціональної моделі, які необхідні для виконання проектування реляційних БД.

Моделі процесів:

- бізнес-модель процесів (ієрархія функцій системи);
- модель потоку даних.

Моделі станів:

- модель життєвого циклу сутності;
- набір специфікацій функцій системи (вимоги);
- опис функцій системи через сутності й атрибути;
- бізнесу-правила, які реалізують функції.

Елементи інформаційної моделі ПО є вхідними даними для завдання створення логічної моделі даних. Елементи функціональної моделі ПО є вхідними даними для завдання проектування додатків БД і, частково, для завдання створення фізичної моделі БД.

1.2.4 Процес проектування БД

Значна частина проектів у сфері інформаційних технологій спрямована на розроблення й створення ІС, у рамках яких здійснюється обробка даних різної складності. Практично у всіх таких проектах вирішується завдання проектування БД певного типу.

В експлуатації БД повинна задовольняти набір вимог щодо ряду інтегрованих параметрів, таких, як:

- функціональність й адаптованість;
- продуктивність обробки трансакцій;
- пропускна здатність;
- час реакції;
- безпека.

Такі параметри іноді перебувають у протиріччі один з одним. Так, високі вимоги до функціональності на даному конкретному устаткуванні можуть вступати у конфлікт з високими вимогами до продуктивності. Наприклад, звіти можуть генеруватися протягом декількох годин і знизити в цей час реакції користувачів, що працюють із системою у діалоговому режимі.

Таким чином, процес проектування БД полягає у досягненні компромісів між функціональними, інформаційними, апаратними, архітектурними й технологічними вимогами до БД і будується на інформованому прийнятті рішень за структурою БД.

Проектування БД - це пошук засобів задоволення функціональних вимог засобами наявної комп'ютерної технології з урахуванням заданих обмежень.

Як правило, IT-проекти із створення БД містять у собі такі етапи:

1. Визначення стратегії побудови системи.
2. Аналіз вимог до БД.
3. Проектування БД.
4. Реалізація БД.
5. Тестування.
6. Впровадження БД.

Етап проектування БД вважається одним із найскладніших етапів створення БД, який не має явно вираженого початку й закінчення. У порівнянні з аналізом вимог до БД або розробленням додатків, проектування БД, на думку багатьох провідних фахівців, є невдало структурованим завданням. Якщо всі етапи створення БД перекриваються один з одним у своїй послідовності, то етап проектування перекривається з усіма іншими етапами. Проектування починається з моменту прийняття стратегічних рішень і триває на етапах реалізації й тестування.

Процес проектування БД охоплює кілька основних сфер:

- проектування об'єктів БД (таблиці, подання, індекси, тригери, збережені процедури, функції, пакети) для подання даних ПО в БД;
- проектування інтерфейсу взаємодії з БД (форми, звіти й т.д.), тобто проектування додатків, які будуть супроводжувати дані в БД і реалізовувати питально-відповідні відношення на цих даних;
- проектування БД під конкретне обчислювальне середовище або інформаційну технологію (архітектура

"клієнт-сервер", паралельні архітектури, розподілене обчислювальне середовище);

- проектування БД під призначення системи (інтелектуальний аналіз даних, OLAP, OLTP і т.д.).

Типова бізнес-модель процесу проектування БД

Процес проектування БД може бути представлений у вигляді моделі бізнес-процесів. Бізнес-модель процесу проектування дозволяє:

- відобразити суб'єктивну думку розробника БД на процес проектування конкретної БД;
- врахувати особливості IT-проекту, у рамках якого проектується БД;
- досить швидко скласти план проектування конкретної БД;
- прорахувати тривалість проектних робіт (створити тимчасову модель проектування).

Розглянемо типову бізнес-модель процесу проектування БД. На рис. 1.11 наведена контекстна діаграма процесу проектування БД.

Як видно з рисунка 1.11, на вхід процесу проектування БД подаються:

- інформаційна модель ПО БД: діаграми "сутність-зв'язок" (ER-діаграми);
- функціональна модель ПО БД: бізнес-модель процесів, діаграми потоку даних (DF-діаграми), діаграми станів, - діаграми життєвих циклів сутностей, специфікації на системи (вимоги), бізнес-правила;
- загальносистемні вимоги й обмеження;
- завдання зворотного впливу.

На виході процесу проектування БД формуються такі результати:

- фізична модель БД, що може бути перетворена у скрипт для створення БД;

- фізична БД;
- специфікація модулів додатків БД;
- план тестування БД.

Продовжуючи функціональну декомпозицію процесу проектування БД, приходимо до діаграми декомпозиції процесу проектування БД першого рівня, яка

Рисунок 1.11 – Контекстна діаграма процесу проектування БД

відбиває основні, найбільш великі професійні завдання (етапи) проектування БД (рис. 1.12).

Такими завданнями (етапами) є:

- збір і аналіз вхідних даних – це початковий етап проектування, на якому здійснюються збір і контроль якості результатів аналізу ПО БД, готується план проектування БД;

- створення логічної моделі БД – це етап, на якому на підставі інформаційної моделі ПО БД створюється логічна структура БД, незалежна від її реалізації;
- створення фізичної моделі БД: внутрішня схема – це етап, на якому на підставі логічної моделі БД створюється фізична структура БД, залежна від її

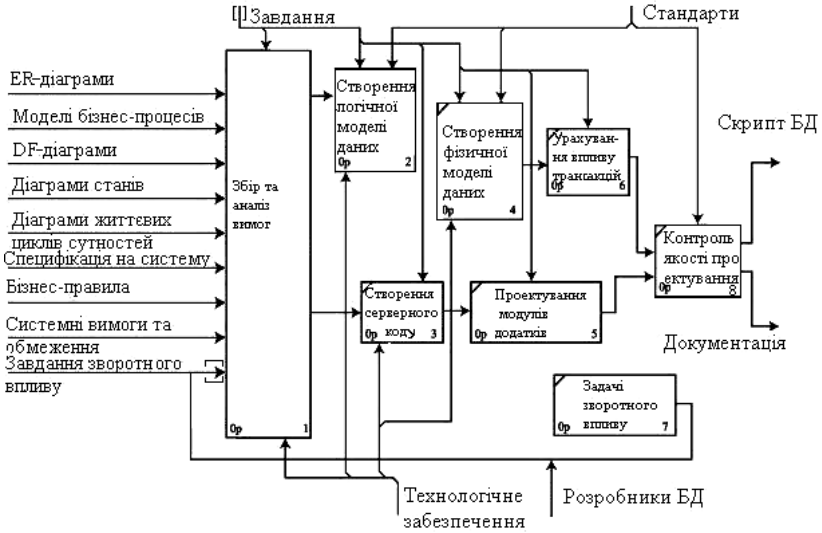


Рисунок 1.12 – Діаграма декомпозиції процесу проектування БД: перший рівень

реалізації. На цьому етапі виконується перетворення відношення логічної моделі реляційної БД у команди створення об'єктів фізичної БД, у результаті чого створюється так звана внутрішня схема БД. Додатково може бути створена так звана зовнішня схема БД, останнє відбиває точку зору користувачів на дані в БД;

- створення фізичної моделі БД: урахування впливу трансакцій – це етап, на якому аналізуються

можливі трансакції системи, за потреби виконується денормалізація відношення для забезпечення більш високої продуктивності БД;

- створення серверного коду – це етап, на якому на підставі функціональної моделі ПО БД створюється серверний код БД у вигляді тригерів, збережених процедур і пакетів. Ці модулі створюються розробником БД і виконуються сервером;
- проектування модулів додатків БД – це етап, на якому створюються специфікації модулів додатків, розробляються стратегії тестування БД і додатків, створюється план тестування додатків БД і готуються тестові дані;
- контроль якості проектування БД полягає в перевірці якості результатів проектування на кожному його етапі;
- урахування завдань зворотного впливу полягає у настроюванні деяких трансакцій до БД і локальному перепроектуванні БД відповідно до вимог, що надходять з інших етапів створення БД.

Короткий розгляд завдань створення серверного коду й підготовки скрипту

Професійне завдання проектування БД - розроблення серверного коду БД - виникає, як правило, у обчислювальному середовищі з багатьма користувачами. У цих системах користувачі спільно використовують обчислювальні ресурси, зокрема ресурси дискової пам'яті й оперативної пам'яті процесора. Обчислювальні ресурси можуть бути сконцентровані в одному місці (централізовані обчислення) або розосереджені в різних вузлах, об'єднаних у комп'ютерну мережу (розподілені обчислення). СКБД у кожному разі покликана координувати й здійснювати доступ користувачів до баз даних і їхніх об'єктів.

Більшість сучасних СКБД підтримують концепцію клієнт-серверної технології для розподілених обчислень. Це означає, що існують концентратори обчислень (названі серверами), на яких виконується найбільший обсяг обчислень із даними (сервери БД), і машини користувачів (клієнти), на яких виконуються додатки користувачів. Додатки формують запити у формі команд SQL до БД, відправляють їхнім серверам БД, одержують запитувані дані й обробляють їх.

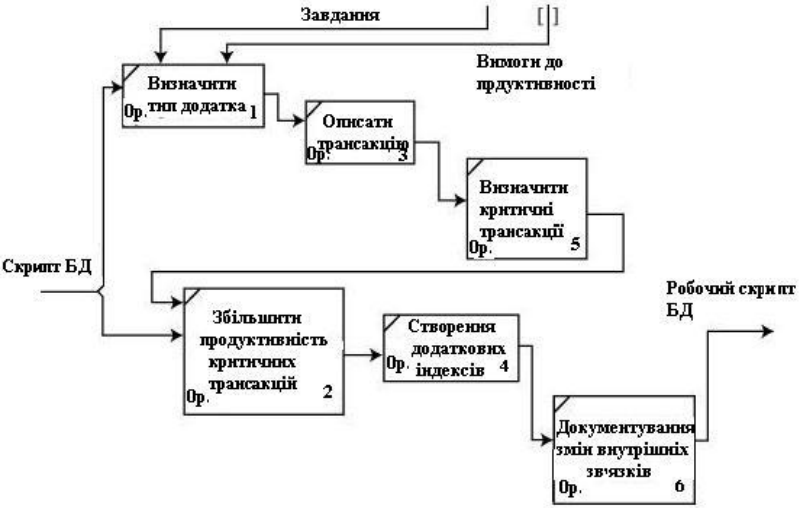


Рисунок 1.13 – Декомпозиція етапу проектування - створення першої ітерації фізичної моделі БД: внутрішня схема

У клієнт-серверному обчислювальному середовищі додаток може взаємодіяти із сервером БД за іншою схемою: коли додаток відправляє запит, цей запит обробляється на сервері, а додатку повертається готовий результат. Робота додатка за другою схемою ґрунтується

на використанні так званого серверного коду (server-side code) - будь-якого коду, виконуваного комп'ютером, на якому встановлена СКБД. Ядро СКБД виконує цей код у БД і повертає додатку тільки результат. Наприклад, це може бути трохи колонок рядка або обчислене значення.

Використання серверного коду може значно скоротити обсяг мережного трафіку і тим самим збільшити продуктивність БД у цілому. Однак СКБД повинна мати вбудовані засоби для розпізнавання й обробки такого коду. Багато фірм-виробників промислових СКБД пропонують процедурні розширення SQL, за допомогою яких можна виконувати порядкову обробку даних, використовувати цикли, складні обчислення й операції керування даними.

Таким чином, розроблення серверного коду зводиться до вирішення таких підзадач:

- ухвалення рішення й створення збережених процедур;
- ухвалення рішення й створення функцій;
- ухвалення рішення й створення пакетів;
- ухвалення рішення й створення тригерів.

Завдання проектування БД - підготовка інсталяційного скрипту для створення БД - деякою мірою завершальна для самостійної роботи розробника БД. Такий скрипт - це один із головних результатів його роботи. Розробник БД, виконавши попередні завдання, фактично виконав свою основну роботу над створенням скрипту для БД.

Завдання створення скрипту БД складається з вирішення великих підзадач:

- створення користувачів, їх ідентифікація й призначення їм привілеїв;
- прив'язка розроблених об'єктів реляційної БД до параметрів фізичного зберігання БД за допомогою створення спеціальних об'єктів БД;

- створення інсталяційного скрипту;
- документування БД.

1.3 Реляційна модель даних

1.3.1 Поняття відношення

Значному поширенню і популярності реляційна модель даних була завдячує двом істотних перевагам:

1) однорідність подання даних у моделі, що обумовлює простоту сприйняття її конструкцій користувачами БД;

2) наявність розвиненої математичної теорії реляційних БД, що обумовлює коректність її застосування.

В основі реляційної моделі даних лежить поняття відношення, яке задається переліком своїх елементів і перерахуванням їх значень. Розглянемо приклад на рис. 1.14. На ньому наведений розклад руху автобусів маршрутом "Москва - Черноголовка - Москва". Бачимо певну структуру. Кожен включений у розклад рейс має свій номер, час відправлення й час у дорозі. Розклад може бути наведено у вигляді таблиці. Заголовки колонок таблиці називаються атрибутами. Перелік їх імен має назви схеми відношення. Кожен атрибут визначає тип даних, що разом з областю його значень називається доменом. Уся таблиця цілком називається відношенням, а кожен рядок таблиці називається кортежем відношення. Таким чином, відношення можна представити у вигляді двовимірної таблиці.

Підходи до визначення поняття відношення можуть бути різними. Математично відношення може бути визначене як безліч кортежів, що є підмножиною декартового добутку фіксованого числа областей (доменів). У результаті одержуємо, що у кожному кортежі

повинне бути однакове число компонентів (атрибутів) і значення кожного з них вибирається з деякого певного домену.

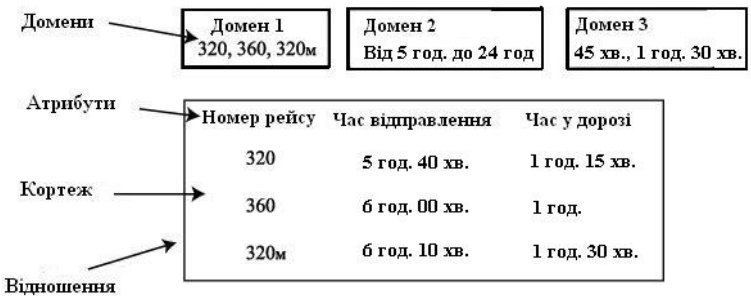


Рисунок 1.14 – Розклад руху автобусів як відношення

Таблична форма подання відношення була введена з метою популяризації моделі серед непідготовлених користувачів БД. Трактування реляційної теорії на рівні таблиць приховують ряд визначень, важливих для розуміння як теорії реляційних БД, так і мови маніпулювання даними.

По-перше, атрибути різних відношень можуть бути визначені на одному домені так само, як і атрибути одного відношення. Це дуже важлива обставина, що дозволяє встановлювати зв'язки за значенням між відношеннями. По-друге, множина математично за своїм визначенням не може мати елементи, що збігаються, і, отже, кортежі у відношенні можна розрізнити лише за значенням їх компонентів. Це теж є дуже важливим для моделі: ніякі два кортежі не можуть мати компонентів, що повністю збігаються. *Таким чином, у реляційній моделі повністю виключається дублювання даних про сутності реального світу*. По-третє, відзначимо, що схема відношення також є

множиною, що дозволяє працювати з ними за допомогою теоретико-множинних операцій. Це є важливим моментом для побудови теорії проектування реляційних схем БД.

Існує певна розбіжність між математичним визначенням відношення й дійсно збереженим відношенням у пам'яті комп'ютера. За визначенням, відношення не може мати два ідентичних кортежі. Однак СКБД, що підтримують реляційну модель даних, зберігають відношення у файлах операційної системи комп'ютера. Розміщення відношень у файлах операційної системи допускає зберігання ідентичних кортежів. Якщо не використовується спеціальна техніка (контроль цілісності щодо первинного ключа), то звичайно більшість промислових СКБД допускають зберігання двох ідентичних кортежів у БД.

Ключем або *ключовим полем* називається унікальне значення, що дозволяє так чи інакше ідентифікувати сутність або частину сутності ПО, тобто ключ - це значення деякого атрибута або атрибутів у кортежі відношення, що представляє екземпляр сутності у реляційній моделі даних.

Прийнято розрізняти первинні ключі й часткові ключі. Математично *первинним ключем* відношення є підмножина звуження декартового добутку, що дозволяє однозначно ідентифікувати кортеж. Якщо первинний ключ містить кілька атрибутів, то він називається складеним ключем, у протилежному разі – атомарним. *Частковим ключем* називається атрибут складеного ключа, якщо він однозначно визначає сукупність неключових атрибутів відношення. Атрибут кортежу, що є первинним ключем іншого відношення, називається *зовнішнім* (іноді стороннім) *ключем*. З визначення відношення випливає така важлива властивість реляційної моделі даних: кожне відношення повинне мати первинний ключ. Зазначимо, що

ключ у контексті моделі ПО БД завжди відображає той або інший ступінь зв'язку між атрибутами сутностей ПО, тобто семантично ключ є засобом моделювання зв'язків у моделі.

Приклад: розглянемо речення "Громадянин Іванов проживав у місті Москві 10 років". Можливими атрибутами у відношенні Місце_проживання є прізвище громадянина, назва міста проживання й час проживання. Прізвище громадянина може виступати як первинний ключ цього відношення, тому що особистість однозначно визначає час її проживання в конкретному місті. Таким чином, щодо цього моделюється зв'язок "проживав" між атрибутами "прізвище" й "місто".

Відношення у реляційній моделі даних, як правило, представляються за допомогою функціональної форми запису (тому що ми записуємо функції декількох змінних у математичному аналізі), при цьому атрибути первинного ключа підкреслюються:

ІМ'Я_ВІДНОШЕННЯ (Атрибути первинного ключа, неключові атрибути).

Приклад. Подання зв'язку відношенням. Представимо зв'язок між особистістю й місцем її проживання через відношення

ПРОЖИВАЄ (Кл. особистість, Кл. населений_пункт, час)
Опис особистості:
ОСОБИСТІСТЬ (Кл. особистість, ПІП/б, вік, стать)
Опис населеного пункту:
НАСЕЛЕНИЙ_ПУНКТ (Кл.населений_пункт, географія, населення)

Однак найбільшого поширення набуло подання відношень у вигляді графічних діаграм, наприклад ER-діаграм, про які ми говорили раніш. Перевагами такого подання є наочність діаграм і можливість їх побудови у ряді CASE-засобів проектування БД.

У підсумку сформулюємо основні властивості реляційної моделі даних, які випливають із поняття відношення як множини:

- усі кортежі одного відношення повинні мати ту саму кількість атрибутів;
- значення кожного з атрибутів повинне належати деякому певному домену;
- кожне відношення повинне мати первинний ключ;
- ніякі два кортежі не можуть мати наборів значень, що повнісю збігаються;
- кожне значення атрибутів повинне бути атомарним, тобто не повинне мати внутрішньої структури й містити як компонент інше відношення;
- реляційна модель даних повинна бути несуперечливою, зокрема, повинні виконуватися: 1) принцип посилальної цілісності - зв'язки між відношеннями повинні бути замкнутими; 2) значення колонок повинні належати тому самому визначеному для них домену;
- порядок проходження кортежів у відношенні не має значення. Порядок є здебільшого властивістю зберігання даних, ніж властивістю безпосередньо самої реляційної моделі даних.

1.3.2 Поняття функціональної залежності в даних

На стадії логічного проектування реляційної БД розробник визначає й вибудовує схеми відношення у рамках деякої ПО, а саме - представляє сутності, групує їх атрибути, виявляє основні зв'язки між сутностями. Так, у найзагальнішому змісті проектування реляційної БД полягає в обґрунтованому виборі конкретних схем відношення з безлічі різних альтернативних варіантів схем.

На практиці побудова логічної моделі БД незалежно від моделі даних виконується з урахуванням двох основних вимог: виключити надмірність і максимально підвищити надійність даних. Ці вимоги випливають з вимоги колективного використання даних групою користувачів.

Тому будь-яке апріорне знання про обмеження ПО, що накладають на взаємозв'язки між даними й значення даних, і знання про їх властивості і взаємини між ними може зіграти певну роль у дотриманні зазначених вище вимог. Формалізація таких апріорних знань про властивості даних ПО БД знайшла своє відображення у концепції функціональної залежності даних, тобто обмежень на можливі взаємозв'язки між даними, які можуть бути поточними значеннями схеми відношень.

Кортежі відношення можуть представляти екземпляри сутності ПО або фіксувати їх взаємозв'язок. Але навіть якщо ці кортежі відповідають схемі відношень й обрані з припустимих доменів, не всякий з них може бути поточним значенням деякого відношення. Наприклад, вік людини рідко буває більше 120 років, або той самий пілот не може одночасно виконувати два різних рейси. Такі обмеження семантики домену практично не впливають на вибір тієї або іншої схеми відношень. Вони являють собою обмеження на типи даних.

Оскільки функціональну залежність можна задати у вигляді таблиці, а таблиця є формою подання відношень, то стає очевидним зв'язок між функціональною залежністю і відношенням. Відношення може задавати функціональну залежність. Це твердження є першою конструктивною ідеєю, яка покладена в основу теорії проектування реляційних БД.

Приклад. Поняття функціональної залежності

Проілюструємо поняття функціональної залежності на прикладі графіка польотів аеропорту.

ГРАФІК_ПОЛЬОТІВ (Пілот, Рейс, Дата_вильоту, Час_вильоту)

Іванов	*100*	*8.07*	*10:20*
Іванов	*102*	*9.07*	*13:30*
Ісаєв	*90*	*7.07*	*6:00*
Ісаєв	*103*	*10.07*	*19:30*
Петров	*100*	*12.07*	*10:20*
Петров	*102*	*11.07*	*13:30*
Фролов	*90*	*8.07*	*6:00*
Фролов	*90*	*12.07*	*6:00*

Відомо, що: кожному рейсу відповідає певний час вильоту; для кожного пілота, дати й часу вильоту можливий тільки один рейс; на певний день і рейс призначається певний пілот.

Отже: "Час_вильоту" функціонально залежить від {"Рейс"}; "Рейс" функціонально залежить від {"Пілот", "Дата_вильоту", "Час_вильоту"}; "Пілот" функціонально залежний від {"Рейс", "Дата_вильоту"}.

Важливим завданням при виявленні функціональних залежностей на атрибутах відношень, що за визначенням є множиною, необхідно з'ясувати, який з атрибутів виступає як аргумент, а який - як значення функціональної залежності. Найбільш доцільними кандидатами в аргументи функціональної залежності є можливі ключі, тому що кортежі представляють екземпляри сутності, які ідентифікуються значеннями атрибутів свого ключа.

1.3.3 Нормальні форми відношень. Створення логічної моделі реляційної БД

Під реляційною БД прийнято розуміти сукупність екземплярів кінцевих відношень. Сукупність схем відношень утворює схему реляційної БД.

Схема реляційної БД є логічною моделлю реляційної БД. На основі інформаційної моделі у процесі проектування створюються логічна й фізична моделі даних. Інформаційна модель даних відбиває потреби системи в даних і зв'язку між даними з погляду їх споживачів - користувачів; логічна модель даних є незалежним логічним поданням даних; фізична модель даних містить визначення всіх реалізованих об'єктів у конкретній БД для конкретної СКБД.

Установлення функціональної залежності й одержання найкращого з погляду мінімальності подання множини функціональних залежностей дозволять побудувати найбільш оптимальний варіант БД, що забезпечує надійність зберігання й обробки даних на основі методів еквівалентних перетворень схем відношень реляційної БД. Процес вирішення такого завдання називається нормалізацією відношень інформаційної моделі ПО й полягає у перетворенні її об'єктів у логічні таблиці БД. Основні вимоги наведені нижче:

- первинні ключі відношень повинні бути мінімальними;
- число відношень БД повинне по можливості давати найменшу надмірність даних – вимога надійності даних;
- число відношень БД не повинне приводити до втрати продуктивності системи;
- дані не повинні бути суперечливими, тобто при виконанні операцій включення, видалення й відновлення

даних їх потенційна суперечливість повинна бути зведена до мінімуму;

- схема відношень БД повинна бути стійкою, здатною адаптуватися до змін при її розширенні додатковими атрибутами – вимога гнучкості структури БД;
- розкид часу реакції на різні запити до БД не повинен бути великим;
- дані повинні правильно відбивати стан ПО БД у кожен конкретний момент часу – вимога актуальності даних.

Створення системи, що одночасно задовольняє всі вищезгадані вимоги, являє собою складну оптимізаційну задачу, що часом не має однозначного вирішення.

Теорія функціональних залежностей дозволяє встановити певні вимоги до схем відношень у реляційній БД. Ці вимоги формулюються у термінах властивостей відношень і називаються *нормальними формами схем відношень*. Кожна нормальна форма відношень пов'язана з певним класом функціональної залежності, які представлені у відношеннях. Одним з очевидних засобів усунення потенційної суперечливості даних у відношеннях логічної моделі реляційної БД є їх розбивка на два або більше відношень, у кожному з яких є наявною тільки одна функціональна залежність.

Процес усунення потенційної суперечливості й надмірності даних у відношеннях реляційної БД називається *нормалізацією вихідних схем відношень*. Нормалізація відношень полягає у виконанні декомпозиції відношень, що перебувають у попередній нормальній формі, на два або більше відношень, які задовольняють вимоги наступної нормальної форми.

У теорії реляційних БД звичайно виділяється така послідовність нормальних форм: перша нормальна форма (1NF); друга нормальна форма (2NF); третя нормальна

форма (3NF); нормальна форма Бойса-Кодда (BCNF); четверта нормальна форма (4NF); п'ята нормальна форма, або нормальна форма проекції-з'єднання (5NF або PJ/NF).

Основні властивості нормальних форм полягають у такому: кожна наступна нормальна форма у деякому змісті краще попередньої нормальної форми; при переході до наступної нормальної форми властивості попередніх нормальних форм зберігаються.

Перша нормальна форма – відношення перебуває у 1NF, якщо всі атрибути відношення є простими (вимога атомарності атрибутів), тобто не мають компонентів. Іншими словами, домен атрибута повинен складатися з неподільних значень і не може містити в собі безліч значень із множини елементарних доменів.

Нехай є змінне відношення: Employer_Project_Task {Em_Number, Em_Degrees, Em_Pay, Pr_Number, Em_Task}. Атрибути містять відповідно дані про номер справи, розряд та заробітну плату службовця, номер проекту й завдання, що виконує службовець у даному проекті. Припустимо, що розряд службовця визначає розмір його заробітної плати, й що кожен службовець може брати участь у декількох проектах за умови виконання тільки одного завдання. Тоді очевидно, що єдино можливим ключем відношення є складений атрибут { Em_Number, Pr_Number }. Діаграма мінімальної множини ER показана на рис. 1.15. У наведеному відношенні деякі функціональні залежності атрибутів від можливого ключа не є мінімальними. Це призводить до так званих аномалій відновлення. Під *аномаліями відновлення* розуміються труднощі, з якими зустрічаються при виконанні операцій додавання кортежів у відношення (INSERT), видалення кортежів (DELETE) і модифікації кортежів (UPDATE).

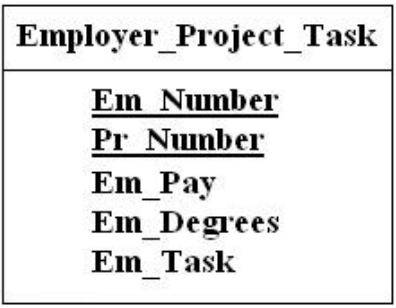


Рисунок 1.15 – ER-діаграма відношення Employer_Project_Task

Стосовно нашого прикладу:

- додавання кортежів – ми не можемо доповнити відношення Employer_Project_Task даними про службовця, який ще не бере участі у жодному проекті (Em_Number є частиною первинного ключа й не може містити невизначених значень). Між іншим часто буває, що спочатку службовця приймають на роботу, встановлюють його розряд і розмір заробітної плати, а лише потім призначають для нього проект;
- видалення кортежів – ми не можемо зберегти у відношенні Employer_Project_Task дані про службовця, який завершив участь у своєму останньому проекті (з тієї причини, що значення атрибута Pr_Number для цього службовця стає невизначеним);
- модифікація кортежів – щоб змінити розряд службовця, ми будемо змушені модифікувати всі кортежі з відповідним значенням атрибута Em_Number. В іншому випадку буде порушений природний зв'язок Em_Number → Em_Degrees (в одного службовця є тільки один розряд).

Для подолання цих труднощів можна зробити декомпозицію змінного відношення Employer_Project_Task на два змінні відношення – Employer {Em_Number,

Em_Degrees, Em_Pay} і Employer_Project_Task {Em_Number, Pr_Number, Em_Task}. На рис. 1.16 показані діаграми ER цих відношень. Тепер ми можемо легко впоратися з операціями відновлення.

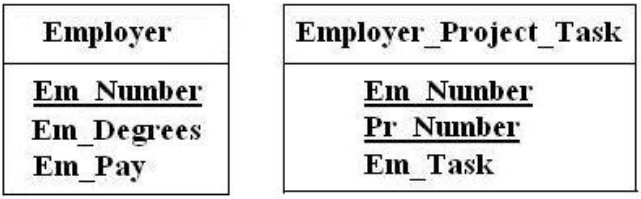


Рисунок 1.16 – ER-діаграми у змінних відношеннях Employer і Employer_Project_Task

Друга нормальна форма

Будемо вважати атрибут відношення ключовим, якщо він є елементом якого-небудь ключа відношення. В іншому випадку атрибут буде вважатися неключовим. Відношення перебуває у 2NF, якщо воно перебуває у 1NF, і всі неключові атрибути відношення функціонально мінімально залежать від первинного ключа. Іншими словами, 2NF вимагає, щоб відношення не містило часткових функціональних залежностей.

Стосовно нашого прикладу: відношення Employer знаходиться у 2NF, а відношення Employer_Project_Task – ні, оскільки атрибут Em_Task функціонально залежить від двох ключових атрибутів: Em_Number та Pr_Number. Будь-яке змінне відношення, що перебуває у 1NF, але не перебуває у 2NF, може бути зведене до набору змінних відношень, що перебувають у 2NF. У результаті декомпозиції ми одержуємо набір проекцій вихідного змінного відношення, природне з'єднання значень яких відтворює значення вихідного змінного відношення (тобто це декомпозиція без втрат).

Третя нормальна форма

Відношення перебуває у 3NF, якщо воно перебуває в 2NF, і всі неключові атрибути відношення залежать тільки від первинного ключа. Іншими словами, 3NF вимагає, щоб відношення не містило транзитивних функціонального зв'язку неключових атрибутів від ключа.

Функціональні залежності відношення Employer, як і раніше, породжує деякі аномалії відновлення. Вони викликаються наявністю транзитивного зв'язку Em_Number → Em_Pay (через зв'язок Em_Number → Em_Degrees і Em_Degrees → Em_Pay). Ці аномалії пов'язані з надмірністю зберігання значення атрибута Em_Pay у кожному кортежі, що характеризує службовців з тим самим розрядом:

- додавання кортежів – неможливо зберегти дані про новий розряд (і відповідному йому розміру зарплати), поки не з'явиться службовець із новим розрядом. Первинний ключ не може містити невизначені значення;
- видалення кортежів – при звільненні останнього службовця з даним розрядом ми втратимо інформацію про наявність такого розряду й відповідний розмір зарплати;
- модифікація кортежів – при зміні розміру зарплати, що відповідає деякому розряду, ми будемо змушені змінити значення атрибута Em_Pay у кортежах усіх службовців, яким призначений цей розряд (інакше не буде виконуватися зв'язок Em_Degrees → Em_Pay).

Можлива декомпозиція: для подолання цих труднощів зробимо декомпозицію змінного відношення Employer на два змінні відношення – Employer1 {Em_Number, Em_Degrees} й Degrees {Em_Degrees, Em_Pay}. На рис. 1.17 показані ER-діаграми цих змінних відношень.

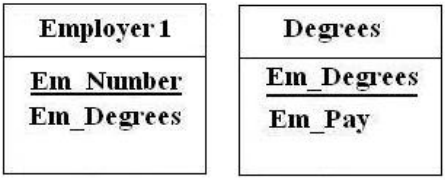


Рисунок 1.17 – ER-діаграми у змінних відношеннях Employer1 і Degrees

Таким чином, процедура зведення відношення до 3NF складається у виконанні двох проекцій: по правій і по лівій частині транзитивного функціонального зв’язку.

Зрозуміло, що в процесі нормалізації декомпозиція відношення на незалежні проекції є кращою. Необхідні й достатні умови незалежності проекцій відношення забезпечує *теорема Риссанена*: проекції r1 й r2 відношення r є незалежними тоді й тільки тоді, коли кожний зв'язок у відношенні r логічно виходить зі зв'язку у r1 й r2; загальні атрибути r1 й r2 утворять можливий ключ хоча б для одного з цих відношень.

Проілюструємо правильність цієї теореми на прикладі декомпозиції відношення Employer. У декомпозиції на проекції Employer1 й Degrees загальний атрибут Em_Degrees є можливим (і первинним) ключем відношення Degrees, а єдиний додатковий зв’язок відношення Employer (Em_Number→Em_Pay) логічно виходить зі зв'язку Em_Number→Em_Degrees і Em_Degrees→Em_Pay, виконуваних для відношення Employer1 й Degrees відповідно.

Атомарним відношенням називається відношення, яке неможливо декомпозувати на незалежні проекції. Далеко не завжди для неатомарних відношень потрібна декомпозиція на атомарні проекції. При виборі способу

декомпозиції необхідно прагнути до одержання незалежних проекцій, але необов'язково атомарних.

Нормальна форма Бойса-Кодда

Наприклад, нехай є змінне відношення Employer_Project_Task1 { Em_Number Em_Name, Pr_Number, Em_Task} з множиною зв'язків, зображених на рис. 1.18.

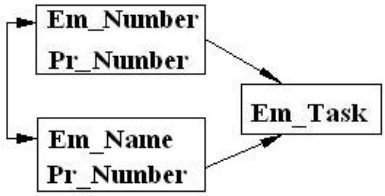


Рисунок 1.18 – Діаграма функціонального зв’язку відношення Employer_Project_Task1

У відношенні Employer_Project_Task1 службовці унікально ідентифікуються як за номерами справ, так і за іменами. Отже, існують зв’язки Em_Number→Em_Name й Em_Name→Em_Number. Але один службовець може брати участь у декількох проектах, тому можливими ключами є {Em_Number, Pr_Number} і {Em_Name, Pr_Number}.

Очевидно, що хоча у відношенні Employer_Project_Task1 усі зв’язки неключових атрибутів від можливих ключів є мінімальними й транзитивні зв’язки відсутні, цьому відношенню властиві аномалії відновлення. Наприклад, у разі зміни імені службовця необхідно обновити атрибут Em_Name в усіх кортежах відношення, що відповідають даному службовцеві. Інакше буде порушений зв'язок Em_Number→Em_Name, і БД виявиться у неузгодженому стані.

Причиною відзначених аномалій є те, що у вимогах 2NF й 3NF не була потрібна мінімальна функціональна

залежність від первинного ключа атрибутів, що є компонентами інших можливих ключів. Проблему вирішує нормальна форма, яку історично прийнято називати нормальною формою Бойса-Кодда і яка є уточненням 3NF у разі наявності декількох можливих ключів, що перекриваються.

Змінна відношення перебуває в нормальній формі Бойса-Кодда (BCNF) у тому і тільки в тому випадку, коли будь-яка виконувана для цього змінного відношення нетривіальнго і мінімального функціонального зв'язку має як детермінант деякий можливий ключ даного відношення.

Відношення Employer_Project_Task1 може бути наведене до BCNF шляхом однієї з двох декомпозицій: Employer_Number_Name {Em_Number, Em_Name} і Employer_Number_Project_Task {Em_Number, Pr_Number, Pr_Task} і Employer_Number_Name {Em_Number, Em_Name} і Employer_Name_Project_Task {Em_Name, Pr_Number, Pr_Task} (зв’язки і значення результуючих змінних відношень наведені на рис. 1.19).

Четверта нормальна форма

Розглянемо ще одну можливу інтерпретацію змінної відношення Employer_Project_Task. Припустимо, що кожен службовець може брати участь у декількох проектах, але в кожному проекті ним повинні виконуватися ті самі завдання. Можливе значення змінної відношення Employer_Project_Task показано на рис. 1.20.

Додавання кортежу – якщо службовець, який вже бере участь у проектах, приєднується до нового проекту, то до тіла значення змінної відношення Employer_Project_Task необхідно додати стільки кортежів, скільки завдань виконує цей службовець.

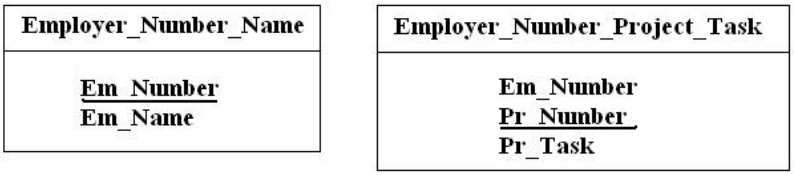


Рисунок 1.19 – ER-діаграми відношень Employer_Number_Name і Employer_Number_Project_Task

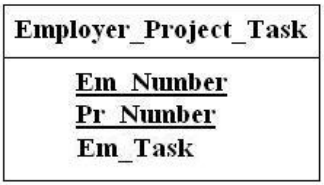


Рисунок 1.20 – Можливе значення змінної відношення Employer_Project_Task

Видалення кортежів – якщо службовець припиняє участь у проектах, то відсутня можливість зберегти дані про завдання, які він може виконувати.

Модифікація кортежів – при зміні одного із завдань службовця необхідно змінити значення атрибута Em_Task у скількох кортежах, у скількох проектах бере участь службовець.

Труднощі, пов'язані з відновленням змінної відношення Employer_Project_Task, вирішуються шляхом його декомпозиції на два змінні відношення: Employer_Project_Number {Em_Number, Pr_Number} і Employer_Task {Em_Number, Em_Task}. Значення цих змінних відношень, що відповідають значенню змінної відношення Employer_Project_Task, показані на рис. 1.21.

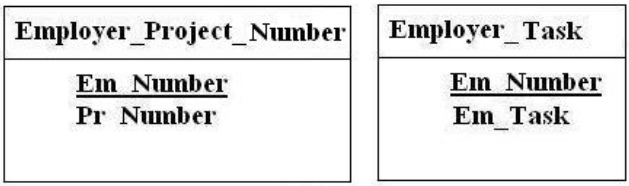


Рисунок 1.21 – Діаграми відношень Employer_Project_Number і Employer_Task

Зверніть увагу, що останній варіант змінної відношення Employer_Project_Task в BCNF, оскільки всі атрибути заголовка відношення входять до складу єдино можливого ключа. Раніше обговорені принципи нормалізації тут незастосовані, але ми одержали корисну декомпозицію. Справа в тому, що у випадку останнього варіанта відношення ми маємо справу з новим видом залежності, уперше виявленим Роном Фейджином у 1971 р. Фейджин назвав залежності цього виду багатозначними (multi-valued dependency - MVD).

У відношенні Employer_Project_Task виконуються дві MVD: Em_Number→Pr_Number і Em_Number→Em_Task. Перша MVD означає, що кожному значенню атрибута Em_Number відповідає обумовлена тільки цим значенням множина значень атрибута Pr_Number. Аналогічно трактується друга MVD.

У змінній відношення r з атрибутами A, B, C (у загальному випадку складовими) є багатозначна залежність B від A (AB) у тому і тільки в тому випадку, коли множина значень атрибута B, що відповідає парі значень атрибутів A і C, залежить від значення A і не залежить від значення C. Багатозначні залежності мають цікаву властивість "подвійності", що демонструє така лема Фейджина:

У відношенні r {A, B, C} виконується MVD A→→B у тому і тільки в тому випадку, коли виконується MVD A→→C.

Функціональний зв'язок є частковим випадком MVD, коли множина значень залежного атрибута обов'язково складається з одного елемента. Таким чином, якщо виконується зв'язок A→B, то виконується й MVD A→→B.

Теорема Фейджина. Нехай є змінна відношення r з атрибутами A, B, C (у загальному випадку складовими). Відношення r декомпозується без втрат на проекції {A, B} і {A, C} тоді й тільки тоді, коли для нього виконується MVD A→→B | C.

Відношення перебуває у 4NF, якщо воно перебуває в 3NF або BCNF, і всі незалежні багатозначні функціональні зв’язки рознесені в окремі відношення з тим самим ключем. Іншими словами, 4NF застосовується за наявності у відношенні більш ніж однієї MVD і вимагає, щоб відношення не містило незалежних багатозначних MVD.

На практиці проектування реляйціних БД методом нормалізації звичайно завершується після досягнення 4NF, і відношення, що перебувають в 4NF, як правило, перебувають і в 5NF. 5NF є "остаточною" нормальною формою, яку можна досягти в процесі нормалізації на основі проекцій.

Нормальні форми характеризуються такими властивостями:

1NF - всі атрибути відношення прості;

2NF - відношення перебуває в 1NF і не містить часткових залежностей;

3NF - відношення перебуває в 2NF і не містить транзитивних залежностей від ключа;

BKNF - відношення перебуває в 3NF і не містить залежностей ключів від неключових атрибутів;

4NF, застосовується за наявності більш ніж однієї багатозначної залежності - відношення перебуває в BKNF або 3NF і не містить незалежних багатозначних функціональних залежностей.

1.3.4 Алгоритм методу декомпозиції відношення

Тепер нам відомо, із чого почати нормалізацію - з універсального відношення; що перевірити - знаходження вихідного відношення в BKNF; що почати - декомпозицію вихідного відношення на два інших відношення; і коли зупинитися - всі відношення БД у BKNF. Таким чином, можна сформулювати загальний алгоритм проектування логічної моделі реляційної БД методом декомпозиції:

Алгоритм методу декомпозиції відношення

1. Розроблення універсального відношення для БД.
2. Визначення всіх функціональних залежностей між атрибутами відношення.
3. Визначення, чи перебуває відношення в BKNF. Якщо так, то завершити проектування; в іншому випадку відношення повинне бути розбите на два інших відношення.
4. Повторення пунктів 2 і 3 для кожного нового відношення, отриманого у результаті декомпозиції.

Правило ланцюжка полягає в такому:

Якщо A>B>C, то як функціональна залежність для здійснення проекції використовується крайня права залежність або "кінець ланцюжка" B>C.

2 Введення в структуровану мову запитів - SQL

2.1 Елементи мови SQL

У даній темі розглянемо елементи мови SQL (Structured Query Language). Мова SQL стала фактично стандартною мовою доступу до БД. Усі СКБД, що претендують на назву "реляційні", реалізують той або інший діалект SQL. Багато які нереляційні системи також мають у цей час засоби доступу до реляційних даних. Метою стандартизації є переносимість додатків між різними СКБД.

Треба відмітити, що в цей час жодна система не реалізує стандарт SQL у повному обсязі. Крім того, в усіх діалектах мови є можливості, що не є стандартними. Таким чином, можна сказати, що кожен діалект - це надмножина деякої підмножини стандарту SQL. Це ускладнює переносимість додатків, розроблених для одних СКБД в інші СКБД.

Мова SQL оперує термінами, що трохи відрізняються від термінів реляційної теорії, наприклад, замість "відношення" використовуються "таблиці", замість "кортежів" - "рядки", замість "атрибутів" - "колонки" або "стовпці".

Стандарт мови SQL хоча й базується на реляційній теорії, але у багатьох місцях відходить він неї. Наприклад, відношення реляційної моделі даних не допускає наявності однакових кортежів, а таблиці у термінології SQL можуть мати однакові рядки. Є й інші відмінності.

Мова SQL є реляційно повною. Це означає, що будь-який оператор реляційної алгебри може бути виражений підходящим оператором SQL.

2.2 Припустимі типи даних

Будь-який діалект SQL підтримують три загальних типи даних: строковий, числовий і тип для подання дати й часу. Завдання типу даних визначає значення й довжину даних, а також формат їхнього подання при візуалізації.

Для всіх типів даних визначено так зване нуль-значення, що вказує на відсутність даних у колонці зазначеного типу, тобто та обставина, що значення даних у цей момент часу невідоме.

Дані строкового типу являють собою послідовність рядків символів. Строкові дані можуть бути задані як з визначеною довжиною (ключові слова char або varchar (довжина рядка)), так і без зазначення довжини (ключове слово long varchar) для подання рядків довільної довжини. Тип даних varchar2 визначає рядок символів змінної довжини, що має максимальний розмір size. На відміну від строкового типу з визначеною довжиною з рядками long varchar не допускаються операції порівняння, і вони не можуть бути використані у виразах і як аргументи більшості вбудованих функцій. Рядки останнього типу можуть застосовуватися для збереження бітових образів. Стандарт SQL-92 не має типу long varchar й varchar.

Зверніть увагу на тип даних varchar2. Він так само, як і тип даних char, призначений для подання алфавітно-цифрових даних. Але він має формат змінної довжини. Останнє означає, що довжина колонки такого типу дорівнює числу символів у ній, у той час як колонка типу char використовує весь виділений для неї простір.

Існують два типи числових даних. Цілі й речовинні значення (наприклад, сальдо банківського рахунку або ставка відсотка). Вони є об'єктом математичної обробки. Строкові числові дані, у яких єдино припустимими символами є цифри (наприклад, номери банківських рахунків).

Числові типи даних призначені для подання цілих чисел, чисел з десятковою крапкою й чисел із плаваючою крапкою. Будь-яке подання чисел задається своєю точністю й масштабом. Точність визначає припустиме подання кількості значущих цифр числа, а масштаб - кількості значущих цифр після десяткової крапки.

Для подання цілих чисел використовуються типи interger (точність 10 значущих цифр) і smallint (точність 5 значущих цифр).

Для подання чисел з фіксованою десятковою крапкою використовуються типи number (точність, масштаб) (для чисел з точністю до 15 значущих цифр) і decimal (точність, масштаб) (для чисел заданої точності до 15 значущих цифр). Якщо вказати для колонки тип number без завдання масштабу, максимальне число значущих цифр для Oracle буде 105. Замість завдання точності й масштабу може бути зазначений символ *. Це буде еквівалентно завданню типу number. Розбіжність між цими типами даних полягає в тому, що для типу number немає необхідності стежити за точністю при виконанні операцій.

Для подання чисел із плаваючою крапкою в SQL передбачені такі типи даних:

Double Precision - для чисел з точністю від 22 до 53 значущих цифр;

Float (точність) - для подання чисел з точністю від 1 до 21 значущої цифри;

Real - для чисел з точністю за замовчуванням (залежить від конкретної реалізації).

Тип даних для подання дати й часу відсутній у стандарті SQL. Звичайно в конкретних діалектах SQL використовуються три типи для подання таких даних:

datestamp (timestamp) - для подання дати й часу;

date - для подання дати;

time - для подання часу.

Константи, вирази, системні змінні.

Константи звичайно визначають єдине значення і, відповідно до типу даних, що представляють, можуть бути строковими, числовими й представляти дату/час. Строкові константи повинні бути взяті в одинарні лапки.

В SQL існує ряд визначених системних змінних, які можна використовувати у виразах замість імен колонок і констант. До таких змінних віднсоять такі:

NULL - для подання невизначених значень;

ROWID - (в SQLBase) внутрішній системний номер рядка в таблиці;

USER - ім'я користувача, активного в цей момент;

SYSDATETIME - системний поточний час і дата;

SYSDATE - системна поточна дата;

SYSTIME - системний поточний час;

SYSTIMEZONE - часовий пояс, установлений у системі.

Виразом в SQL є ітем або комбінація ітемів з припустимими для них операціями, що дає єдине значення. Як ітеми можуть виступати імена колонок, константи, зв'язані змінні, результати обчислень функцій, системні змінні й інші вирази. При цьому якщо один з ітемів має нуль-значення, то результат виразу також має нуль-значення.

2.3 Оператори SQL

Основу мови SQL становлять оператори, умовно розбиті на кілька груп за виконуваними функціями.

Можна виділити такі групи операторів (перелічені не всі оператори SQL):

Оператори DDL (Data Definition Language) - оператори визначення об'єктів БД

- CREATE SCHEMA - створити схему БД;

- DROP SHEMA - видалити схему БД;
- CREATE TABLE - створити таблицю;
- ALTER TABLE - змінити таблицю;
- DROP TABLE - видалити таблицю;
- CREATE DOMAIN - створити домен;
- ALTER DOMAIN - змінити домен;
- DROP DOMAIN - видалити домен;
- CREATE COLLATION - створити послідовність;
- DROP COLLATION - видалити послідовність;
- CREATE VIEW - створити подання;
- DROP VIEW - видалити подання.

Оператори DML (Data Manipulation Language) - оператори маніпулювання даними:

- SELECT - відібрати рядок з таблиць;
- INSERT - додати рядок в таблицю;
- UPDATE - змінити рядок в таблиці;
- DELETE - видалити ряодк в таблиці;
- COMMIT - зафіксувати внесені зміни;
- ROLLBACK - відкотити внесені зміни.

Оператори захисту й керування даними:

- CREATE ASSERTION - створити обмеження
- DROP ASSERTION - видалити обмеження
- GRANT - надати привілею користувачеві або додатку на маніпулювання об'єктами;
- REVOKE - скасувати привілей користувача або додатка.

Крім того, є групи операторів установки параметрів сеансу, одержання інформації про БД, оператори статичного SQL, оператори динамічного SQL.

Найбільш важливими для користувача є оператори маніпулювання даними (DML).

Приклади використання операторів маніпулювання даними

INSERT - вставка рядків у таблицю:

Приклад 1. Вставка одного рядка в таблицю:

```
INSERT INTO
  P (PNUM, PNAME)
  VALUES (4, "Іванов");
```

Приклад 2. Вставка в таблицю декількох рядків, обраних з іншої таблиці (у таблицю TMP_TABLE вставляються дані про постачальників з таблиці P, що мають номери, більші 2):

```
INSERT INTO
  TMP_TABLE (PNUM, PNAME)
  SELECT PNUM, PNAME
    FROM P
    WHERE P.PNUM>2;
```

UPDATE - відновлення рядків у таблиці

Приклад 3. Відновлення декількох рядків у таблиці:

```
UPDATE P
  SET PNAME = "Пушников"
  WHERE P.PNUM = 1;
```

DELETE - видалення рядків із таблиці

Приклад 4. Видалення декількох рядків у таблиці:

```
DELETE FROM P
  WHERE P.PNUM = 1;
```

Приклад 5. Видалення всіх рядків у таблиці:

```
DELETE FROM P;
```

Приклади використання оператора SELECT

Оператор SELECT є фактично найважливішим для користувача і найскладнішим оператором SQL. Він призначений для вибірки даних з таблиць, тобто він, що властиво, і реалізує одне з основних призначень БД - надавати інформацію користувачеві.

Оператор SELECT завжди виконується над деякими таблицями, що входять у БД.

Насправді в БД можуть бути не тільки постійно збережені таблиці, а також тимчасові таблиці й так звані

подання. Подання - це той, що просто зберігається в БД, SELECT-вираз. З погляду користувачів подання - це таблиця, яка не зберігається постійно в БД, а "виникає" у момент звертання до неї. З погляду оператора SELECT і постійно збережені таблиці, і тимчасові таблиці й подання мають зовсім однаковий вигляд. Звичайно, при реальному виконанні оператора SELECT системою враховуються розбіжності між збереженими таблицями й поданнями, але ці розбіжності сховані від користувача.

Результатом виконання оператора SELECT завжди є таблиця. Таким чином, за результатами дій оператор SELECT схожий на оператори реляційної алгебри. Будь-який оператор реляційної алгебри може бути виражений певним чином сформульованим оператором SELECT. Складність оператора SELECT визначається тим, що він містить у собі всі можливості реляційної алгебри, а також додаткові можливості, яких у реляційній алгебрі немає.

Відбір даних з однієї таблиці

Приклад. Вибрати всі дані з таблиці постачальників (ключові слова ***SELECT…FROM***…):

```
SELECT*
FROM P;
```

У результаті одержимо нову таблицю, що містить повну копію даних з вихідної таблиці P.

Приклад. Вибрати всі рядки з таблиці постачальників, що задовольняють деяку умову (ключове слово ***WHERE***…):

```
SELECT*
FROM P
WHERE P...PNUM>2;
```

Як умову в розділі WHERE можна використовувати складні логічні вирази, що використовують поля таблиць, константи, порівняння (>, <, = і т.д.), дужки, сполучники AND і OR, заперечення NOT.

Приклад. Вибрати деякі колонки з вихідної таблиці (зазначення списку колонок, що відбираються):

```
SELECT P.NAME
  FROM P;
```

У результаті одержимо таблицю з однією колонкою, що містить усі найменування постачальників. Якщо у вихідній таблиці були присутні кілька постачальників з різними номерами, але однаковими найменуваннями, то в результуючій таблиці будуть рядки з повтореннями - дублікати рядків автоматично не відкидаються.

Приклад. Вибрати деякі колонки з вихідної таблиці, видаливши з результату повторювані рядки (ключове слово ***DISTINCT***):

```
SELECT DISTINCT P.NAME
  FROM P;
```

Використання ключового слова DISTINCT приводить до того, що у результуючій таблиці будуть вилучені всі повторювані рядки.

Приклад. Використання скалярних виразів і перейменувань колонок у запитах (ключове слово ***AS***…):

```
SELECT TOVAR...TNAME, TOVAR.KOL,
            TOVAR.PRICE, "="AS EQU,
            TOVAR.KOL*TOVAR.PRICE AS SUMMA
FROM TOVAR;
```

У результаті одержимо таблицю з колонками, яких не було у вихідній таблиці TOVAR:

TNAME	**KOL**	**PRICE**	**EQU**	**SUMMA**
Болт	10	100	=	1000
Гайка	20	200	=	4000
Гвинт	30	300	=	9000

Приклад. Упорядкування результатів запиту (ключове слово ***ORDER BY***…):

```
SELECT PD...PNUM, PD.DNUM, PD.VOLUME
FROM PD
ORDER BY DNUM;
```

У результаті одержимо таблицю, впорядковану по полю DNUM:

PNUM	DNUM	VOLUME
1	1	100
2	1	150
3	1	1000
1	2	200
2	2	250
1	3	300

Приклад. Упорядкування результатів запиту за декількома полями зі зростанням або спаданням (ключові слова ***ASC***, ***DESC***):

```
SELECT   PD.PNUM, PD.DNUM, PD.VOLUME
FROM PD
ORDER BY  DNUM ASC, VOLUME DESC;
```

У результаті одержимо таблицю, у якій рядки йдуть у порядку зростання значення поля DNUM, а рядка з однаковим значенням DNUM ідуть у порядку спаданням значення поля VOLUME:

PNUM	DNUM	VOLUME
3	1	1000
2	1	150

1	1	100
2	2	250
1	2	200
1	3	300

Якщо явно не зазначені ключові слова ASC або DESC, то за замовчуванням приймається упорядкування за зростанням (ASC).

Відбір даних з декількох таблиць

Приклад. Природне з'єднання таблиць (спосіб 1 - явна вказівка умов з'єднання):

```
SELECT   P.PNUM, P.PNAME, PD.DNUM,
           PD.VOLUME
FROM P, PD
WHERE P.PNUM = PD.PNUM;
```

У результаті одержимо нову таблицю, у якій рядки з даними про постачальників з'єднані з рядками з даними про поставки деталей:

PNUM	**PNAME**	**DNUM**	**VOLUME**
1	Іванов	1	100
1	Іванов	2	200
1	Іванов	3	300
2	Петров	1	150
2	Петров	2	250
3	Сидоров	1	1000

Таблиці, що з'єднують, перелічені у розділі FROM оператора, умова з'єднання наведена у розділі WHERE.

Розділ WHERE, крім умови з'єднання таблиць, може також містити й умови відбору рядків.

Приклад. Природне з'єднання таблиць (спосіб 2 - ключові слова ***JOIN…USING…***):

```
SELECT PNUM, P.PNAME, PD.DNUM,
            PD.VOLUME
FROM P JOIN PD USING PNUM;
```

Ключове слово USING дозволяє явно вказати, за якими із загальних колонок таблиць буде вироблятися з'єднання.

Приклад. Природне з'єднання таблиць (спосіб 3 - ключове слово ***NATURAL JOIN***):

```
SELECT   P.PNUM, P.PNAME, PD.DNUM,
             PD.VOLUME
FROM P NATURAL JOIN PD;
```

У розділі FROM не зазначено, за якими полями виробляється з'єднання. NATURAL JOIN автоматично з'єднує за всіма однаковими полями у таблицях.

Приклад. Природне з'єднання трьох таблиць:

```
SELECT   P.PNAME, D.DNAME, PD.VOLUME
FROM   P NATURAL JOIN PD NATURAL JOIN D;
```

У результаті одержимо таку таблицю:

PNAME	**DNAME**	**VOLUME**
Іванов	Болт	100
Іванов	Гайка	200
Іванов	Гвинт	300
Петров	Болт	150
Петров	Гайка	250
Сидоров	Болт	1000

Приклад. Прямий добуток таблиць:

```
SELECT   P.PNUM, P.PNAME, D.DNUM,
         D.DNAME
FROM P, D;
```

У результаті одержимо таку таблицю:

PNUM	PNAME	DNUM	DNAME
1	Іванов	1	Болт
1	Іванов	2	Гайка
1	Іванов	3	Гвинт
2	Петров	1	Болт
2	Петров	2	Гайка
2	Петров	3	Гвинт
3	Сідоров	1	Болт
3	Сідоров	2	Гайка
3	Сидоров	3	Гвинт

Оскільки не зазначена умова з'єднання таблиць, то кожен рядок першої таблиці з'єднується з кожним рядком другої таблиці.

Приклад. З'єднання таблиць за довільною умовою. Розглянемо таблиці постачальників і деталей, якими привласнений деякий статус:

Таблиця 2.1 – Відношення P (Постачальники)

PNUM	PNAME	PSTATUS
1	Іванов	4
2	Петров	1
3	Сидоров	2

Таблиця 2.2 – Відношення D (Деталі)

DNUM	DNAME	DSTATUS
1	Болт	3
2	Гайка	2
3	Гвинт	1

Відповідь на запитання "які постачальники мають право поставляти які деталі?", дає такий запит:

```
SELECT  P.PNUM, P.PNAME,
        P.PSTATUS,
        D.DNUM, D.DNAME, D.DSTATUS
FROM P, D
WHERE P.PSTATUS >= D.DSTATUS;
```

У результаті одержимо таку таблицю:

PNUM	PNAME	PSTATUS	DNUM	DNAME	DSTATUS
1	Іванов	4	1	Болт	3
1	Іванов	4	2	Гайка	2
1	Іванов	4	3	Гвинт	1
2	Петров	1	3	Гвинт	1
3	Сидоров	2	2	Гайка	2
3	Сидоров	2	3	Гвинт	1

2.4 Використання імен кореляції (аліасів, псевдонімів)

Іноді доводиться виконувати запити, у яких таблиця з'єднується сама із собою або одна таблиця з'єднується

двічі з іншою таблицею. При цьому використовуються імена кореляції (аліаси, псевдоніми), які дозволяють розрізняти копії та таблиці-оригінали. Імена кореляції вводяться у розділі FROM і йдуть через пробіл після імені таблиці. Імена кореляції повинні використовуватися як префікс перед ім'ям стовпця й відокремлюються від імені стовпця крапкою. Якщо у запиті вказуються ті самі поля з різних екземплярів однієї таблиці, вони повинні бути перейменовані для усунення неоднозначності в іменах колонок результуючої таблиці. Визначення імені кореляції діє тільки під час виконання запиту.

Приклад. Відібрати всі пари постачальників таким чином, щоб перший постачальник у парі мав статус, більший од статусу другого постачальника:

```
SELECT  P1.PNAME AS PNAME1,
        P1.PSTATUS AS PSTATUS1,
        P2.PNAME AS PNAME2,
        P2.PSTATUS AS PSTATUS2
FROM    P P1, P P2
WHERE P1.PSTATUS1 > P2.PSTATUS2;
```

У результаті одержимо таку таблицю:

PNAME1	PSTATUS1	PNAME2	PSTATUS2
Іванов	4	Петров	1
Іванов	4	Сидоров	2
Сидоров	2	Петров	1

Приклад. Розглянемо ситуацію, коли деякі постачальники (назвемо їх контрагенти) можуть виступати як постачальники деталей, так як і одержувачі. Таблиці, що зберігають дані, можуть мати такий вигляд:

Таблиця 2.3 – Відношення CONTRAGENTS

Номер контрагента NUM	**Найменування контрагента NAME**
1	Іванов
2	Петров
3	Сидоров

Таблиця 2.4 – Відношення DETAILS

Номер деталі DNUM	**Найменування деталі DNAME**
1	Болт
2	Гайка
3	Гвинт

Таблиця 2.5 – Відношення CD (Поставки)

Номер постачальника PNUM	**Номер одержувача CNUM**	**Номер деталі DNUM**	**Кількість, що поставляється, VOLUME**
1	2	1	100
1	3	2	200
1	3	3	300
2	3	1	150
2	3	2	250
3	1	1	1000

У таблиці CD поля PNUM й CNUM є зовнішніми ключами, що посилаються на потенційний ключ NUM у таблиці CONTRAGENTS.

Відповідь на запитання, "хто кому що у якій кількості поставляє", дається таким запитом:

```
SELECT  P.NAME AS PNAME,
            C.NAME AS CNAME,
            DETAILS.DNAME, CD.VOLUME
FROM    CONTRAGENTS P, CONTRAGENTS C,
            DETAILS, CD
WHERE  P.NUM = CD.PNUM AND
            C.NUM = CD.CNUM AND
            D.DNUM = CD.DNUM;
```

У результаті одержимо таку таблицю:

PNAME	CNAME	DNAME	VOLUME
Іванов	Петров	Болт	100
Іванов	Сидоров	Гайка	200
Іванов	Сидоров	Гвинт	300
Петров	Сидоров	Болт	150
Петров	Сидоров	Гайка	250
Сидоров	Іванов	Болт	1000

Цей самий запит може бути виражений дуже великою кількістю способів, наприклад, так:

```
SELECT  P.NAME AS PNAME,
            C.NAME AS CNAME,
            DETAILS.DNAME, CD.VOLUME
FROM    CONTRAGENTS P, CONTRAGENTS C,
            DETAILS NATURAL JOIN CD
WHERE  P.NUM = CD.PNUM AND
            C.NUM = CD.CNUM;
```

2.5 Вбудовані функції

Арифметичні функції

SQL підтримує повний набір арифметичних операцій і математичних функцій для побудови арифметичних виразів над колонками БД (+, -, *, /, ABS, LN, SQRT і т.д.). Перелік основних вбудованих математичних функцій поданий нижче:

ABS(X) Повертає абсолютне значення числа X

ACOS(X) Повертає арккосинус числа X

ASIN(X) Повертає арксинус числа X

ATAN(X) Повертає арктангенс числа X

COS(X) Повертає косинус числа X

EXP(X) Повертає експоненту числа X

SIGN(X) Повертає -1, якщо X<0,0, якщо X=0, +1, якщо X>0

LN(X) Повертає натуральний логарифм числа X

MOD(X,Y) Повертає залишок від розподілу X на Y

CEIL(X) Повертає найменше ціле, більше або таке, що дорівнює X

ROUND(X,n) Округлює число X до числа з n знаками після крапки

SIN(X) Повертає синус числа X

SQRT(X) Повертає квадратний корінь числа X

TAN(X) Повертає тангенс числа X

FLOOR(X) Повертає найбільше ціле менше або таке, що дорівнює X

LOG(a,X) Повертає логарифм числа X на основі A

SINH(X) Повертає гіперболічний синус числа X

COSH(X) Повертає гіперболічний косинус числа X

TANH(X) Повертає гіперболічний тангенс числа X

TRANC(X,n) Зменшує число X до числа з n знаками після десяткової крапки

POWER(A,X)Повертає значення A, піднесе до степеня X

Арифметичні вирази необхідні для одержання даних, які безпосередньо не зберігаються в колонках таблиць БД, але значення яких необхідні користувачеві. Припустимо, що вам необхідний список службовців, який показує виплату, що одержав кожен службовець із урахуванням премій і штрафів.

Функції для обробки дати

У діалекті SQL є невеликий набір функцій для маніпулювання колонками з типом date. Список основних функцій обробки дати й часу наведений нижче:

SYSDATE Повертає поточну дату й час

ROUND(D[,F]) Округлює значення дати D відповідно до заданого шаблону

TRANC(D[,F]) Зменшує значення дати D відповідно до заданого шаблону

NEXT_DAY(D,S) Повертає дату дня, що є першим днем, більш пізнім, ніж поточна дата із назвою S

Якщо вам був потрібний список нових службовців, що з'явились за останній квартал в організації, то ви можете написати запит у такому видгляді:

```
SELECT ENAME, HIREDATE,
            HIREDATE + 92  DAYS
FROM    EMPLOYEE
WHERE HIREDATE + 92 DAYS > SYSDATE
          AND DEPNO=30;
```

Ключове слово SYSDATE завжди повертає поточну дату. У цьому прикладі також показано, як використовується арифметичний оператор додавання зі змінними типу "дата". До змінного типу "дата" можна додавати й віднімати від нього ціле число днів, місяців,

років, годин, хвилин, секунд, мікросекунд. Для цього використовуються відповідні ключові слова (DAY, MONTH і т.д.), що випливають за цілою константою (дробова частина ігнорується, якщо ви вказуєте число з десятковою крапкою). Є обмеження на використання дужок у таких виразах (так, висновок у дужках виразу 1 DAYS + 1 YEARS призведе до помилки).

Використання агрегатних функцій у запитах

У мові SQL передбачені такі оператори агрегатних функцій:

AVG(X) = AVG(ALL X) AVG(DISTINCT X) Обчислює середнє значення аргументу, що може бути виразом будь-якого типу. Нуль-значення ігноруються, ключове слово DISTINCT пригнічує дублікати.

COUNT(*) COUNT(X) = COUNT(ALL X) COUNT(DISTINCT X) Обчислює числа ітемів. При вказівці * завжди повертається число рядків у таблиці. Вказівка DISTINCT пригнічує дублікати.

MAX(X) = MAX(ALL X) MAX (DISTINCT X) Обчислює максимальне значення аргументу, що може бути виразом будь-якого типу. Нуль-значення ігноруються, ключове слово DISTINCT пригнічує дублікати.

MIN(X) = MIN(ALL X) MIN (DISTINCT X) Обчислює мінімальне значення аргументу, що може бути виразом будь-якого типу. Нуль-значення ігноруються, ключове слово DISTINCT пригнічує дублікати.

SUM(X) = SUM(ALL X) SUM (DISTINCT X) Обчислює суму значення аргументу, що може бути виразом будь-якого типу. Нуль-значення ігноруються, ключове слово DISTINCT пригнічує дублікати.

STDDEV([DISTINCT|ALL]X) Обчислює стандартне відхилення на безлічі значень аргументу, що може бути виразом будь-якого типу. Нуль-значення ігноруються, ключове слово DISTINCT пригнічує дублікати.

VARIANCE([DISTINCT|ALL]) Обчислює квадрат дисперсії.

Приклад. Одержати загальну кількість постачальників (ключове слово ***COUNT***):

```
SELECT COUNT(*) AS N
FROM P;
```

У результаті одержимо таблицю з одним стовпцем й одним рядком, що містить кількість рядків з таблиці P:

Приклад. Одержати загальну, максимальну, мінімальну й середню кількості деталей, що поставляються, (ключові слова SUM**,** MAX**,** MIN**,** AVG):

```
SELECT   SUM(PD.VOLUME) AS SM,
         MAX(PD.VOLUME) AS MX,
         MIN(PD.VOLUME) AS MN,
         AVG(PD.VOLUME) AS AV
FROM PD;
```

У результаті одержимо таку таблицю з одним рядком:

SM	MX	MN	AV
2000	1000	100	333. 33333333

Використання агрегатних функцій з угрупуваннями

Приклад. Для кожної деталі одержати сумарну кількість, що поставляється (ключове слово ***GROUP BY***…):

```
SELECT..DNUM, SUM(PD.VOLUME) AS SM
GROUP BY PD.DNUM;
```

Цей запит буде виконуватися у такий спосіб. Спочатку рядки вихідної таблиці будуть згруповані так, щоб у кожну групу потрапили рядки з однаковими значеннями DNUM. Потім усередині кожної групи буде

просумовано поле VOLUME. Від кожної групи до результуючої таблиці буде включений один рядок.

До переліку полів оператора SELECT, який містить розділ GROUP BY, можна включати тільки агрегатні функції й поля, які входять в умову групування. Наступний запит видасть синтаксичну помилку:

```
SELECT  PD.PNUM, PD.DNUM,
                SUM(PD.VOLUME) AS SM
GROUP BY PD.DNUM;
```

Причина помилки у тому, що у перелік полів, які відбираються, включене поле PNUM, що не входить у розділ GROUP BY. І дійсно, у кожну отриману групу рядків може входити кілька рядків з різними значеннями поля PNUM. З кожної групи рядків буде сформовано по одному підсумковому рядку. При цьому немає однозначної відповіді на питання, яке значення вибрати для поля PNUM у підсумковому рядку.

Деякі діалекти SQL не вважають це за помилку. Запит буде виконаний, але передбачити, які значення будуть внесені у поле PNUM у результуючій таблиці, неможливо.

Приклад. Одержати номери деталей, сумарна кількість поставки яких перевищує 400 (ключове слово ***HAVING***…):

Умова, що сумарна кількість поставки повинна бути більше 400, не може бути сформульована у розділі WHERE, тому що в цьому розділі не можна використовувати агрегатні функції. Умови, що використовують агрегатні функції, повинні бути розміщені у спеціальному розділі HAVING:

```
SELECT   PD.DNUM, SUM(PD.VOLUME) AS SM
GROUP BY PD.DNUM
HAVING SUM(PD.VOLUME) > 400;
```

В одному запиті можуть зустрітися як умови відбору рядків у розділі WHERE, так і умови відбору груп у розділі HAVING. Умови відбору груп не можна перенести з розділу HAVING у розділ WHERE. Аналогічно й умови відбору рядків не можна перенести з розділу WHERE у розділ HAVING, за винятком умов, що включають поля зі списку угрупування GROUP BY.

2.6 Використання підзапитів

Дуже зручним засобом, що дозволяє формулювати запити більш зрозумілим чином, є можливість використання підзапитів, вкладених в основний запит.

Приклад. Одержати список постачальників, статус яких менше максимального статусу у таблиці постачальників (порівняння з підзапитом):

```
SELECT *
FROM P
WHERE P.STATYS <  (SELECT MAX(P.STATUS)
                   FROM P;
```

Оскільки поле P.STATUS порівнюється з результатом підзапиту, то підзапит повинен бути сформульований так, щоб повертати таблицю, що складається рівно з одного рядка й однієї колонки.

Результат виконання запиту буде еквівалентний результату такої послідовності дій:

1. Виконати один раз вкладений підзапит і одержати максимальне значення статусу.

2. Просканувати таблицю постачальників P, щоразу порівнюючи значення статусу постачальника з результатом підзапиту, і відібрати тільки ті рядки, у яких статус менше максимального.

Приклад. Використання предиката ***IN***. Одержати перелік постачальників, що поставляють деталь номер 2:

```
SELECT *
FROM P
WHERE P.PNUM IN (SELECT DISTINCT PD.PNUM
                 FROM PD WHERE PD.DNUM = 2);
```

У цьому випадку вкладений підзапит може повертати таблицю, що містить кілька рядків.

Результат виконання запиту буде еквівалентний результату такої послідовності дій:

1. Виконати один раз вкладений підзапит й одержати список номерів постачальників, що поставляють деталь номер 2.

2. Просканувати таблицю постачальників P, щоразу перевіряючи, чи втримується номер постачальника в результаті підзапиту.

Приклад. Використання предиката ***EXIST***. Одержати перелік постачальників, що поставляють деталь номер 2:

```
SELECT *
FROM P
WHERE EXIST (SELECT *
  FROM PD
  WHERE PD.PNUM = P.PNUM AND PD.DNUM = 2);
```

Результат виконання запиту буде еквівалентний результату такої послідовності дій:

1. Просканувати таблицю постачальників P, щоразу виконуючи підзапит із новим значенням номера постачальника, узятим з таблиці P.

2. У результат запиту включити тільки ті рядки з таблиці постачальників, для яких вкладений підзапит повернув непорожню множину рядків.

На відміну від двох попередніх прикладів вкладений підзапит містить параметр (зовнішнє посилання), переданий з основного запиту - номер постачальника P.PNUM. Такі підзапити називаються

корельованими (***correlated***). Зовнішнє посилання може набирати різних значень для кожного рядка-кандидата, оцінюваного за допомогою підзапиту, тому підзапит повинен виконуватися заново для кожного рядка, який відбирається в основному запиті. Такі підзапити характерні для предиката EXIST, але можуть бути використані і в інших підзапитах.

Може здатися, що запити, які містять корельовані підзапити будуть виконуватися повільніше, ніж запити з некорельованими підзапитами. Насправді це не так, тому що те, як користувач сформулював запит, не визначає, як цей запит буде виконуватися. Мова SQL є не процедурною, а декларативною. Це означає, що користувач, який формулює запит, просто описує, яким повинен бути результат запиту, а як цей результат буде отриманий - за це відповідає сама СКБД.

Приклад. Використання предиката ***NOT EXIST***. Одержати перелік постачальників, що не поставляють деталь номер 2:

```
SELECT *
FROM P
WHERE NOT EXIST (SELECT *
  FROM PD
  WHERE PD.PNUM = P.PNUM AND PD.DNUM = 2);
```

Також як і у попередньому прикладі, тут використовується корельований підзапит. Відмінність полягає у тому, що в основному запиті будуть відібрані ті рядки з таблиці постачальників, для яких вкладений підзапит не видасть ні одного рядка.

Приклад. Одержати імена постачальників, що поставляють всі деталі:

```
SELECT DISTINCT PNAME  FROM P
WHERE NOT EXIST (SELECT *
  FROM D
```

```
WHERE NOT EXIST  (SELECT *
      FROM PD
WHERE  PD.DNUM = D.DNUM AND
            PD.PNUM = P.PNUM));
```

Даний запит містить два вкладених підзапити й реалізує реляційну операцію розподілу відношень.

Сам внутрішній підзапит параметризований двома параметрами (D.DNUM, P.PNUM) і має такий зміст: відібрати всі рядки, що містять дані про поставки постачальника з номером PNUM деталі з номером DNUM. Заперечення NOT EXIST свідчить про те, що даний постачальник не поставляє дану деталь. Зовнішній до нього підзапит сам є вкладеним і параметризованим параметром P.PNUM, має зміст: відібрати перелік деталей, які не поставляються постачальником PNUM. Заперечення NOT EXIST свідчить про те, що для постачальника з номером PNUM не повинне бути деталей, які не поставлялися б цим постачальником. Це в точності означає, що в зовнішньому запиті відбираються тільки постачальники, що поставляють всі деталі.

2.7 Використання об'єднання, перетину й різниці

Приклад. Одержати імена постачальників, що мають статус, більший 3, або що поставляють хоча б одну деталь номер 2 (об'єднання двох підзапитів - ключове слово ***UNION***):

```
SELECT P.PNAME
FROM P
WHERE P.STATUS > 3  UNION SELECT P.PNAME
  FROM P, PD
  WHERE P.PNUM = PD.PNUM AND
    PD.DNUM = 2;
```

Результуючі таблиці поєднуваних запитів повинні бути сумісні, тобто мати однакову кількість стовпців і

однакові типи стовпців у порядку їхнього перерахування. Не потрібно, щоб поєднувані таблиці мали однакові імена колонок. Це відрізняє операцію об'єднання запитів у SQL від операції об'єднання у реляційній алгебрі. Найменування колонок у результуючому запиті будуть автоматично взяті з результату першого запиту в об'єднанні.

Приклад. Одержати імена постачальників, що мають статус, більший 3, і одночасно поставляють хоча б одну деталь номер 2 (перетинання двох підзапитів - ключове слово ***INTERSECT***):

```
SELECT P.PNAME
FROM P
WHERE P.STATUS > 3 INTERSECT
  SELECT P.PNAME
  FROM P, PD
  WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2;
```

Приклад. Одержати імена постачальників, що мають статус, більший 3, за винятком тих, хто поставляє хоча б одну деталь номер 2 (різниця двох підзапитів - ключове слово ***EXCEPT***):

```
SELECT P.PNAME
FROM P
WHERE P.STATUS > 3 EXCEPT
  SELECT P.PNAME
   ROM P, PD
  WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2;
```

3 Реалізація доступу до БД у середовищі DELPHI

3.1 Механізми доступу до БД

VCL-бібліотека класів середовища проектування Delphi надає ряд класів, що дозволяють швидко й ефективно розробляти різні додатки БД. Ці класи представлені наступними групами:

- компоненти для доступу до даних, реалізуючі: доступ через машину БД BDE (Borland Database Engine), який надає доступ через ODBC-драйвери або через внутрішні драйвери машини БД BDE (компоненти сторінки BDE-палітри інструментів); доступ через ADO-об'єкти (Active Data Objects), в основі якого лежить застосування технології OLE DB (компоненти сторінки ADO); доступ до локального або вилученого SQL-сервера InterBase (компоненти сторінки InterBase); доступ за допомогою легковагих драйверів dbExpress; доступ до БД при багатоланковій архітектурі (компоненти сторінки DataSnap);
- візуальні компоненти, що реалізують інтерфейс користувача;
- компоненти для зв'язку джерел даних з візуальними компонентами, які надають інтерфейс користувача;
- компоненти для візуального проектування звітів.

Основними механізмами доступу до даних, підтримуваним в Delphi, є:

- ODBC - доступ через ODBC-драйвери БД або BDE-драйвери;
- OLE DB - доступ з використанням провайдерів даних (OLE DB - це метод доступу до будь-яких даних через стандартний COM-інтерфейс);

- засоби dbExpress, що використовують легковагі драйвери БД;
- засоби доступу до розподілених наборів даних у багатоланковій архітектурі.

Найпростіший механізм керування даними, що використовують ODBC-драйвери, може бути реалізований за наступною схемою:

У модуль даних (або у форму) додається компонент набору даних (об'єкт класу TDataSet) і встановлюється зв'язок із джерелом даних, обумовлене властивістю DatabaseName. Зв'язок може бути зазначена одним із трьох способів: по імені БД, каталогу або псевдоніму (спосіб вказівки зв'язки може бути обмежений типом джерела даних).

У модуль даних (або у форму) додається компонент джерела даних (TDataSourse), що є центральною сполучною ланкою між набором даних й елементами керування. Властивість DataSet компонента типу TDataSourse указує набір даних, формована компонентами таких класів як TTable або TQuery. Якщо компоненти набору даних і джерела даних розташовані в модулі даних, то їх варто додати в проект (команда меню File | Use unit).

У форму додаються елементи керування для роботи з даними, такі як TDBGrid, TDBEdit, TDBCheckbox. Вони зв'язуються з компонентом джерелом даних, що вказується властивістю DataSource. Ім'я поля набору даних визначається властивістю DataField.

Графічно схему роботи з базами даних для дволанкових архітектур у середовищі Delphi можна представити в такий спосіб (рис. 3.1).

Для збереження даних із БД в XML-форматі або двійковому форматі, і назад, для формування набору даних з XML або двійкового файлу застосовується провайдер даних.

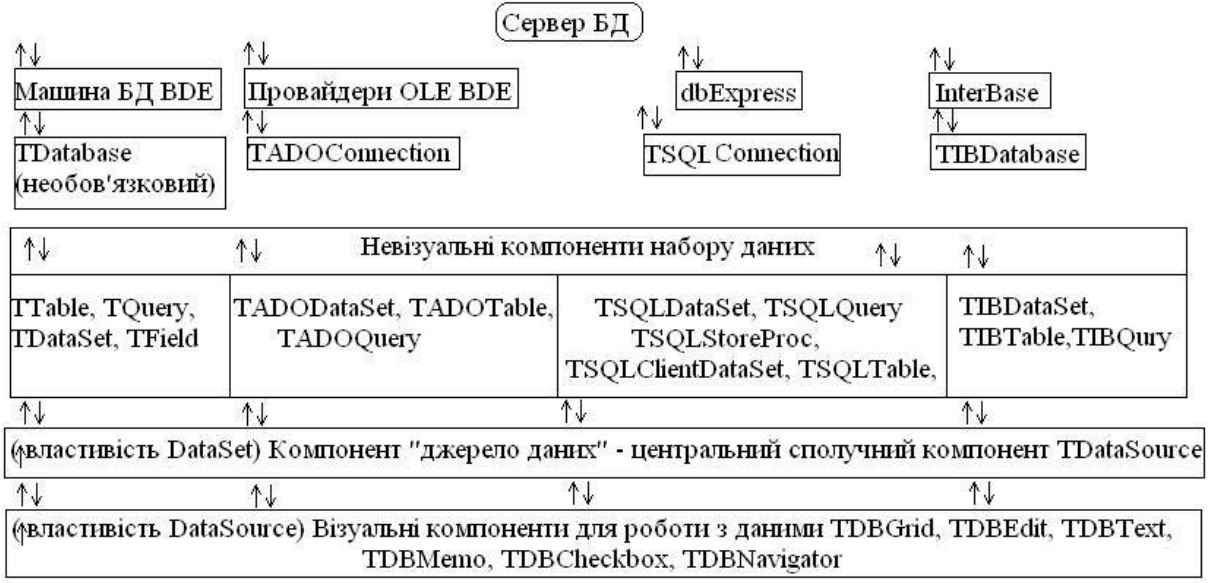


Рисунок 3.1 – Схема роботи з БД у середовищі Delphi

3.2 Набори даних

Базою всіх класів наборів даних є клас TDataSet. Він визначає основу структури всіх наборів даних - масив компонентів типу TField (кожен елемент масиву відповідає стовпцю таблиці).

Набір даних - це впорядкована послідовність рядків, витягнутих із джерела даних. Кожен рядок набору даних складається з полів, що вказують у властивостях класу.

Залежно від механізму доступу, який використовується додатком, базовими класами набору даних можуть бути:

TTable, TQuery, TStoredProc - для одноланкових або дволанкових додатків, які використовують машину БД BDE. Клас TQuery додатково дозволяє виконувати параметричні запити;

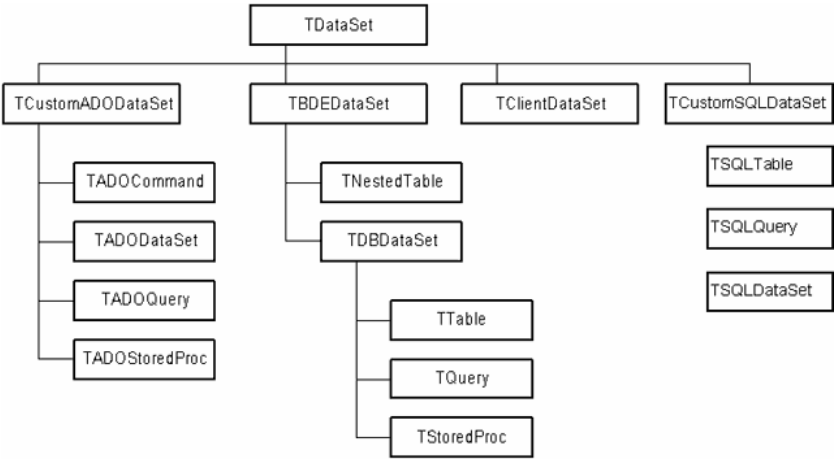
TClientDataSet - для реалізації клієнтського набору даних і для багатоланкової архітектури, яка використовує розподілений доступ;

TADODataSet - для додатків, які використовують ADO-об'єкти;

TSQLDataSet - для доступу до БД за допомогою dbExpress. Цей клас реалізує спрямований набір даних, що функціонує за принципом курсору. Для такого набору даних не створюється кеш пам'яті на клієнті, і серед методів доступу можливі тільки методи Next й First. Редагування записів у спрямованому наборі даних можливо тільки явним виконанням SQL-оператора UPDATE або при установці з'єднання з клієнтським набором даних через провайдера;

TSQLTable й TSQLQuery - для доступу до БД за допомогою dbExpress.

На наступній схемі наведена ієрархія класів наборів дані бібліотеки VCL:



Для визначення набору даних необхідно задати наступні властивості:

- для класу TTable - значення властивостей DatabaseName й TableName;
- для класу TQuery - значення властивості SQL й, можливо, властивості DatabaseName.

Для того щоб читати дані з таблиць або записувати їх у таблиці, набір даних попередньо повинен бути відкритий. Відкрити набір даних можна одним з наступних способів: установити значення властивості Active набору даних рівним True під час виконання додатка (наприклад, Table1.Active:= True;) або в режимі проектування в інспекторі об'єктів; викликати метод Open (наприклад, Table1.Open;).

Аналогічно, закрити набір даних можна викликом методу Close або встановивши значення властивості Active рівним False. Для компонента типу TQuery метод Open може бути виконаний тільки для закритого набору даних: спроба відкрити вже відкритий набір даних ініціює помилку.

Відкриття набору даних спричиняє: ініціацію подій BeforeOpen й AfterOpen; установку стану набору даних в dsBrowse; відкриття курсору для набору даних. Якщо в момент відкриття набору даних відбулася помилка, то стан набору даних установлюється в dsInactive, а курсор закривається.

При роботі з компонентами наборів даних можна обійтися без явного використання компонентів, які реалізують з'єднання з БД. Однак деякі можливості, такі як керування транзакціями або кешировані відновлення, неможливі без компонентів типу TDatabase або TADOConnection. Компонент "БД" TDatabase застосовується для з'єднання із джерелом даних через драйвери BDE або зовнішні ODBC-драйвери. Компонент TADOConnection використовується для створення об'єкта "з'єднання" при доступі через OLE DB, що інкапсулюються за допомогою ADO-об'єктів VCL-бібліотеки.

За замовчуванням при переході від одного запису набору даних до іншої відбувається запис всіх зроблених змін у БД. Для того щоб можна було скасовувати зроблені зміни або виконувати відновлення декількох записів, застосовують кешировані відновлення. Вони дозволяють значно знизити мережний трафік за рахунок того, що всі зроблені зміни зберігаються у внутрішньому кеші й при переході від одного запису до іншого інформація у БД не передається. Щоб включити режим кешированого відновлення, слід встановити значення властивості CachedUpdates рівним True для компонента набору даних. Для присвоєння кэшированного відновлення викликається метод ApplyUpdates, а для скасування - CancelUpdates.

3.3 Класи бібліотеки VCL

Клас TDATASET є базовим для всіх класів наборів даних, які успадковують загальні властивості й методи цього класу, включаючи наступні:

- Active - властивість, що визначає, чи відкритий набір даних;
- CurrentRecord - властивість, що визначає номер поточного запису набору даних;
- DataSource - властивість, що вказує батьківську таблицю (для таблиць, зв'язаних відношенням батьківська - дочірня);
- Bof - властивість, що визначає, чи перебуває курсор на першому записі набору даних;
- Eof - властивість, що визначає, чи досягнуть кінець набору даних;
- FieldCount - властивість, що вказує кількість полів у наборі даних;
- Bookmark - властивість, що вказує поточну закладку в наборі даних. Закладка відзначає позицію в наборі даних. Використовуючи методи TDataSet.GetBookmark й TDataSet.GotoBookmark, додаток може запам'ятовувати й швидко переходити на потрібну позицію в наборі даних;

- Fields - властивість, що представляє собою масив полів набору даних і використовуване для доступу до цих полів. Властивість Fields дозволяє одержати ім'я поля в поточній структурі запису:

```
var  S: String;
begin
S := Fields[0].FieldName;// Ім'я першого поля
S := Fields[1].FieldName;// Ім'я другого поля
...
end;
```

записати в змінну значення поля.

```
var  s: String; i: Integer; d: TDateTime;
s := Fields[0].AsString;
i := Fields[0].AsInteger;
d := Fields[0].AsDate;
```

- Filter - властивість, у яке заноситься рядок, що визначає фільтр для набору даних. Фільтр визначає умова, якій повинні задовольняти доступні записи.

Визначення фільтра повинне задовольняти наступним правилам:

фільтр складається з умов для полів набору даних, об'єднаних логічними операціями AND й OR. Наприклад: F2 > 10 AND F2 <50;

якщо ім'я поля містить пробіли, то воно повинне бути укладене у квадратні дужки або подвійні лапки. Наприклад: [Field Name1] > 50;

Filtered - властивість, що вказує, чи використовується фільтр, заданий властивістю Filter;

Found - властивість, що визначає, чи успішно виконаний пошук методами FindFirst, FindLast, FindNext або FindPrior;

Modified - властивість, що визначає, чи був змінений активний запис;

RecordCount - властивість, що містить загальне число записів у наборі даних;

- State - властивість, що вказує поточний стан набору даних. Ця властивість може приймати наступні значення:

dsInactive - набір даних закритий;

dsBrowse - дані доступні тільки для перегляду;

dsEdi - можна змінювати активний запис;

dsInsert - активним записом є новий запис, поки не збережена;

dsSetKey - перегляд обмеженої безлічі записів (SetRange) або пошук запису;

dsCalcFields - виконується оброблювач події OnCalcFields;

dsFilter - виконується оброблювач події OnFilterRecord;

dsOpening - набір даних перебуває в процесі відкриття.

- Append - метод, що додає в кінець набору даних новий запис;
- Delete - метод, що видаляє поточний запис із БД. Якщо в момент видалення запису набір даних перебуває в неактивному стані, то ініціюється виключення;
- Edit - метод, що переводить поточний запис у режим редагування;
- Cancel - метод, що скасовує зміни, зроблені в поточному записі;
- Post - метод, що виконує внесення змін у БД;
- Refresh - метод, що виконує відновлення результуючого набору шляхом повторного добування даних із БД;
- Insert - метод, що вставляє в набір даних новий запис;
- InsertRecord - метод, що вставляє в набір даних новий запис зі значеннями, які зазначені параметрами методу;
- Close - метод, що закриває набір даних;
- Open - метод, що відкриває набір даних;
- First - метод, що встановлює курсор на перший запис набору даних і робить цей запис активним;
- Last - метод, що встановлює курсор на останній запис набору даних і робить цей запис активним;
- Next - метод, що переміщає курсор на наступний запис набору даних і робить цей запис активним;

- Prior - метод, що переміщає курсор на попередній запис набору даних і робить цей запис активним.

Клас TDATASOURCE реалізує зв'язок між компонентами - наборами даних й елементами керування, які використовуються для відображення даних.

При побудові відношення між таблицями "батьківська-дочірня" компонентів "джерело даних" служить для зв'язування наборів даних, указуючи батьківський набір даних.

Клас TDataSource містить набір властивостей і методів, що використовуються для доступу до набору даних, включаючи наступні:

- AutoEdit - властивість, що визначає, чи буде автоматично викликатися метод Edit набору даних при одержанні фокуса елементом керування, асоційованим із джерелом даних;
- DataSet - властивість, що вказує використовуваний набір даних.

Змінюючи значення властивості DataSet під час виконання, можна ефективно перемикатися на роботу з різними наборами даних, відображаючи різні набори даних у тих самих елементах керування.

DataSource.DataSet := Table1;.

- Enabled - властивість, що визначає, чи буде елемент керування відображати асоційовані з ним дані, або буде відображатися порожнім;
- State - властивість, що дозволяє визначити стан використовуваного набору даних.

```
if DataSource1.Dataset <> nil then
  //Кнопка доступна тільки в тому випадку, якщо набір
  //даних перебуває в стані редагування
  //або вставки нового запису
```

BtnPost1.Enabled := DataSource1.State in [dsEdit, dsInsert];

Клас TTABLE використовується для доступу до БД за допомогою визначення джерела даних DSN й імені таблиці БД. При цьому допускається вибір всіх полів таблиці або тільки частини полів, а також завдання фільтра, що визначає, які рядка таблиці будуть доступні.

Компоненти типу TTable можуть використовувати всі властивості й методи, наслідувані від класу TDataSet, а також властивості й методи класу TTable для набору даних, включаючи наступні:

- DatabaseName - властивість, що визначає ім'я джерела даних DSN;
- CanModify - властивість, що визначає, чи може додаток виконувати вставку, редагування й видалення записів у таблиці;
- DefaultIndex - властивість, що визначає, чи належні дані в таблиці бути впорядковані при її відкритті. Якщо значення властивості дорівнює True (за замовчуванням), то виконується впорядкування по первинному ключі або унікальному індексі;
- IndexName - властивість, що дозволяє визначити вторинний індекс, використовуваний для сортування набору даних, які відкриваються;
- Exclusive - властивість, що дозволяє встановити винятковий режим доступу до таблиці (значення властивості повинне бути визначене до відкриття таблиці);
- MasterSource - властивість, що визначає ім'я компонента "джерело даних" батьківської таблиці для встановлення відносини між таблицями "батьківська-дочірня";
- MasterFields - властивість, що визначає одне або кілька полів з батьківської таблиці, службовців для зв'язку з відповідними полями даної дочірньої таблиці (Це задає

відношення між батьківською й дочірньою таблицями. Поля в списку розділяються крапкою з комою);

- ReadOnly - властивість, що дозволяє встановити для таблиці режим доступу "тільки для читання";
- TableName - властивість, що вказує використовувану таблицю БД;
- RecNo - властивість, що вказує номер поточного запису набору даних;
- FindKey - метод, що виконує пошук значення або значень, перерахованих у списку, для ключового поля;
- FindNearest - метод, що переміщає курсор на запис, що містить значення, найбільш близьке до зазначеного значення ключового поля (пошук може виконуватися як по одному значенню, так і по декількох, якщо використовується складений індекс).
- Locate - метод, використовуваний для пошуку першого входження значення зазначеного поля або набору полів (якщо запис знайдений, то вона стає поточною).

Клас TQUERY дозволяє виконувати будь-який SQL-оператор, припустимий по синтаксису ODBC-драйвером. Якщо як здійсненний оператор використовується SQL-оператор SELECT, то компонент повертає набір даних (результуючий набір). На відміну від класу Ttable, клас TQuery дозволяє створювати набори даних з декількох таблиць, а також обмежувати одержуваний набір даних певними умовами. Це скасовує необхідність добування всіх записів таблиці в набір даних, що, у свою чергу, заощаджує пам'ять, скорочує мережний трафік для вилучених БД і зменшує час доступу.

Для визначення набору даних TQuery варто встановити значення властивості SQL й, можливо, властивості DatabaseName (властивість DatabaseName визначає ім'я джерела даних, але для деяких БД можна задати повне ім'я таблиці, що включає місце розташування

файлу, у тексті SQL-оператора, - у цьому випадку властивість DatabaseName не використовується). Найбільш правильним підходом все-таки варто вважати той, при якому ім'я DSN джерела даних указується властивістю DatabaseName, а в SQL-операторі визначається тільки ім'я таблиці без визначення її місця розташування.

За замовчуванням, набір даних, формований компонентом типу TQuery, не редагується. Для того щоб значення в створеному наборі даних можна було редагувати, необхідно виконати одне з наступних дій:

зв'язати компонент TQuery з компонентом типу TUpdateSQL (наприклад: Query1. Query1.UpdateObject:= UpdateSQL1;) і визначити для останнього значення властивості ModifySQL (наприклад: update TBL1 set F1 = :F1, F2 = :F2 where F3 = :OLD_F3);

установити значення властивості RequestLive рівним True (підтримка цієї можливості залежить від використовуваної БД).

- Клас TQuery містить властивості й методи, використовувані для роботи з набором даних, включаючи наступні:

- DataSource - властивість, що дозволяє вказати батьківський набір даних (для відношення "батьківський-дочірній").

Наприклад, якщо властивість SQL містить значення 'SELECT * FROM Tbl1 t WHERE (t.FNo = :FNo)', те значення змінного зв'язку :FNo буде визначатися із джерела даних, зазначеного властивістю DataSource.

- Params - властивість, що містить список параметрів для SQL-оператора.

- RequestLive - властивість, що визначає, чи буде можливість редагувати створюваний набір даних (можливість одержання що модифікує результуючого набору залежить від використовуваного SQL-сервера);

- SQL - властивість, що містить текст SQL-оператора (для автоматичного формування SQL-оператора можна викликати з контекстного меню компонента TQuery діалог SQL Builder);
- DatabaseName - властивість, що визначає ім'я джерела даних, які підключаються, (ім'я DSN джерела даних або ім'я, уведене класом типу TDatabase);
- ExecSQL - метод, що виконує SQL-оператор, зазначений властивістю SQL (для SQL-оператора, що створює набір даних, замість ExecSQL використовується метод Open).
- ExecSQL можна викликати для таких SQL-операторів як INSERT, UPDATE, DELETE, CREATE TABLE і т.п.

Якщо перед викликом ExecSQL не був викликаний метод Prepare, то SQL-оператор буде одночасно й відкомпільований, і виконаний.

- Prepare - метод, що виконує компіляцію SQL-оператора.

Виклик цього методу перед ExecSQL збільшує швидкість виконання запиту при багаторазовому повторенні викликів ExecSQL для того самого оператора (наприклад, параметризованого запиту). Це дозволяє відкомпілювати SQL-оператор тільки один раз, а потім багаторазово його виконувати.

Клас TSQLTABLE представляє таблицю БД, доступну для клієнта як спрямований набір даних. Такий набір містить всі записи для полів, певних у класі TSQLTable. Об'єкт типу TSQLTable повинен бути пов'язаний з об'єктом типу TSQLConnection, що визначає з'єднання із джерелом даних. Для відображення такого набору даних не можна використовувати таблицю, тому що в клієнта відсутній кеш пам'яті для набору даних. Значення полів таблиці можна відображати компонентами

типу TDBText або TDBEdit. Для переміщень по наборі записів доступні тільки методи First й Next.

Після розміщення в модулі даних або на формі компонента треба виконати наступні дії: установити значення властивості SQLConnection компонента TSQLTable, вибравши доданий раніше компонент типу TSQLConnection із пропонованого списку; визначити ім'я таблиці БД, використовуваної для побудови набору даних, визначивши властивість TableName компонента TSQLTable.

Клас TUPDATESQL дозволяє для наборів даних, створених з доступом "тільки для читання", підтримувати можливість їхнього відновлення за допомогою виконання SQL-оператора.

Клас TUpdateSQL реалізує наступні властивості й методи:

- DeleteSQL - властивість, що визначає SQL-оператор DELETE.
- InsertSQL - властивість, що визначає SQL-оператор INSERT.
- ModifySQL - властивість, що визначає SQL-оператор UPDATE.
- ExecSQL - метод, що виконує один із заданих SQL-операторів (залежно від значення параметра, що вказує наступними константами: ukDelete, ukInsert,ukModify).

Клас TDATABASE реалізує роботу з об'єктом "БД" і надає засоби контролю над з'єднанням з БД. Компонент типу TDatabase дозволяє управляти транзакціями.

Для роботи з компонентом TDatabase необхідно встановити значення властивостей AliasName й DatabaseName. Якщо значенням властивості AliasName зазначений DSN існуючого джерела даних, то розроблювач може сам визначити будь-який внутрішній (для додатка)

псевдонім БД і задати його у властивості DatabaseName. У цьому випадку для будь-якого набору даних у списку значень властивості DatabaseName буде відображатися поряд з усіма доступними DSN джерелами даних і внутрішній псевдонім, заданий властивістю DatabaseName компонента TDatabase.

У тому випадку, якщо DSN не визначений, то властивість DatabaseName повинна містити повне ім'я файлу БД, а властивість DriverName - указувати використовуваний ODBC-драйвер.

Компонент типу TDatabase дозволяє управляти режимами роботи з наборами даних і транзакціями, використовуючи наступні властивості й методи:

- Exclusive - властивість, що дозволяє додатку одержати винятковий доступ до БД (якщо це підтримується SQL-сервером);
- InTransaction - властивість, що вказує, чи була виконана для БД виклик методу StartTransaction.
- ReadOnly - метод, що вказує, чи встановлений для з'єднання з БД доступ "тільки читання".
- TransIsolation - метод, що задає рівень ізоляції при керуванні транзакціями. Рівень ізоляції визначає, як дана транзакція буде взаємодіяти з іншими транзакціями, що працюють із тими самими таблицями. Властивість TransIsolation може бути зазначено одним з наступних значень:

tiDirtyRead - транзакція може читати дані, які були змінені іншою транзакцією, але для яких не був виконаний виклик Commit (фіксація змін);

tiReadCommitted - дозволяє в одній транзакції читати фіксовані зміни, зроблені в базі даних іншою транзакцією;

tiRepeatableRead - істинність даних гарантується на увесь час читання, і транзакція не бачить ніяких змін,

зроблених іншою транзакцією. Прочитаний запис залишається постійної, поки в ній не будуть зроблені зміни усередині самої транзакції;

- StartTransaction - метод, що відкриває нову транзакцію;
- Commit - метод, що виконує фіксацію поточної транзакції;
- Rollback - метод, що виконує відкіт поточної транзакції;
- Execute - метод, що виконує зазначений параметром SQL-оператор, що не повертає результуючого набору.

Клас TADOCONNECTION забезпечує з'єднання з даними, доступ до яких реалізується через ADO-об'єкти. ADO-об'єкти дозволяють працювати з різними сховищами даних, які можуть і не бути SQL-операторами. Об'єкти типу TADOConnection використовують для доступу до даних OLE DB провайдери.

Компоненти TADOCommand й TADODataSet зв'язуються із джерелом даних за допомогою об'єкта TADOConnection, указуючи посилання на нього як значення властивості Connection.

Для ідентифікації з'єднання необхідно визначити значення властивості ConnectionString (рядок з'єднання) компонента TADOConnection, що може ґрунтуватися на вказівці: datalink-файлу; рядка з'єднання.

Якщо як значення властивості ConnectionString зазначене ім'я UDL-файлу, то настроювання з'єднання можна виконувати автономно від додатка (наприклад, указуючи ім'я БД Microsoft SQL Server на поточному ПК).

3.4 Класи компонентів керування даними

Компоненти керування даними розташовані на сторінці Data Controls палітри компонентів. Багато хто із

цих компонентів аналогічні елементам керування сторінки Standard, з тією лише відмінністю, що зв'язано через джерело даних (компонент типу TDataSource) з певним полем (або полями) з набору даних (компонентів типу TTable або TQuery).

Бібліотека VCL надає наступні класи компонентів керування даними:

- TDBGrid - клас, що дозволяє відображати запису набору даних у вигляді таблиці й управляти цими записами.
- TDBNavigator - клас, що надає засоби навігації по наборі даних, а також можливості додавання нових записів, включення режиму редагування, присвоєння й скасування зроблених змін. Для того щоб програмно ініціювати дія, виконувана по щиглику на кнопці навігатора, варто викликати метод BtnClick: DBNavigator1.BtnClick(nbPost); // Присвоєння зроблених змін.

Компонент TDBNavigator може відображати кнопки, що вказують наступними константами:

nbFirst - перехід до першого запису;

nbPrior - перехід до попереднього запису;

nbNext - перехід до наступного запису;

nbLast - перехід до останнього запису;

nbInsert - вставка перед поточним записом нового запису й перехід на неї;

nbDelete - видалення поточного запису;

nbEdit - перехід у режим редагування поточного запису;

nbPost - внесення змін поточного запису в БД;

nbCancel - скасування змін, зроблених у поточному записі;

nbRefresh - повторне зчитування значень полів із джерела даних.

- TDBText - клас, що дозволяє як напис відображати значення поля поточного запису набору даних.
- TDBEdit - клас, що реалізує роботу з однорядковим полем редагування.
- TDBMemo - клас, що реалізує багаторядкове поле редагування, у якому можна відображати й змінювати значення поля набору даних.
- TDBImage - клас, що реалізує об'єкт "рисунок", у якому можна відображати й змінювати значення поля набору даних формату BLOB.
- TDBRadioGroup - клас, що реалізує групу радіокнопок, які пов'язані з полем БД. Застосування такого об'єкта надає користувачеві зручну можливість встановлювати значення поля БД, вибираючи його із пропонованих опцій.
- TDBCheckBox - клас, що реалізує компонент "прапорець", що пов'язаний з полем БД.
- TDBListBox - клас, що реалізує компонент "список", який використовується для відображення значень поля набору даних. Значення, відображувані в списку, утримуються у властивості Items.
- TDBComboBox - клас, що реалізує компонент " список, щорозкривається,", що використовується для відображення значень поля набору даних. Значення, відображувані в списку, утримуються у властивості Items.
- TDBLookupListBox - клас, що дозволяє виконувати перегляд списку, заповненого значеннями полів з іншого набору даних. Набір даних, що переглядає, указується властивістю ListSource, що переглядає поле (або поля) - властивістю ListField. Властивість KeyField указує поле набору, що переглядає, даному, відповідному полю поточного набору даних, що вказує властивостями DataField й DataSource. Даний клас дозволяє вибирати

значення поля поточного набору даних з іншого набору даних, що переглядає;

- TDBCtrlGrid - клас, що реалізує особливий вид таблиці, у якій кожен запис відображається на окремій панелі (кількість панелей у компоненті вказується значенням властивості RowCount).

3.5 Події, які ініціюються для наборів даних:

AfterCancel й BeforeCancel - відбувається після/до скасування в додатку всіх змін, зроблених для поточного запису.

AfterClose й BeforeClose - відбувається після/до закриття набору даних і перекладу БД у стан dsInactive.

AfterDelete й BeforeDelete - ініціюється після/до видалення додатком поточного запису, перекладу набору даних у стан dsBrowse і переміщення позиції курсору на попередній запис.

AfterEdit й BeforeEdi - відбувається після/до початку редагування додатком поточного запису.

AfterInsert й BeforeInsert - відбувається після/перше ніж додаток вставить новий запис.

AfterOpen й BeforeOpen - відбувається після/перше ніж додаток відкриє набір даних, але до того, як які-небудь доступні дані будуть відображені.

AfterPost й BeforePost - відбувається до завершення переносу значень активного запису в БД або внутрішній кеш.

AfterRefresh й BeforeRefresh - відбувається після/до відновлення набору даних.

AfterScroll й BeforeScroll - відбувається після/до переміщення позиції курсору на інший запис.

OnCalcFields - відбувається при відкритті набору даних, перекладу його в стан dsEdit, переміщенні фокуса уведення від одного компонента до іншому або від одного

стовпця до іншого, при змінах запису або при добуванні запису з БД, але тільки в тому випадку, якщо значення властивості AutoCalcFields дорівнює True;

OnDeleteError - ініціюється, якщо при спробі видалення рядка відбулася помилка - було кинуте виключення.

OnEditError - ініціюється, якщо при спробі зміни або вставки запису відбулася помилка - було кинуте виключення.

OnPostError - ініціюється, якщо при спробі передати зміну або вставку нового запису відбувається помилка - кидається виключення.

OnFilterRecord - відбувається при зміні активного запису й тільки в тому випадку, якщо властивість State набору даних установлено рівним dsFilter, а властивість Filtered дорівнює True. Щоб запис був включений у набір даних, для неї варто встановити параметр Accept рівним True.

OnNewRecord - відбувається при вставці або додаванні нового запису.

4 Проектування модулів додатків

4.1 Аналіз функціональної моделі предметної області БД

Вхідними даними для вирішення завдання проектування модулів додатків БД є ієрархія функцій. На виході розробник повинен отримати опис (специфікацію) модулів додатків, а в процесі проектування модулів розробник будує відображення бізнес - вимог у специфікації модулів.

Алгоритм дій розробника БД полягає у наступному: спочатку розробник намагається сформулювати бізнес -

вимоги (функції) у самому загальному вигляді, а потім виконує декомпозицію кожної такої бізнес-функції доти, поки не буде отримана деяка функція, яку можна вважати атомарною функцією.

Розглянемо методологію проектування на прикладі. Розглянемо фрагмент ієрархії функцій для обробки заяв про виплату страхового відшкодування. На спрощеній схемі рис. 4.1 показана функція "2. Обробити заяву". Виконання цієї функції включає виконання чотирьох функцій наступного рівня: "2.1. Зареєструвати заяву", "2.2. Ухвалити рішення щодо заяви", "2.3. Здійснити платіж за заявою", "2.4. Закрити заяву".

На рис. 4.1 показана подальша декомпозиція функції "2.2. Ухвалити рішення щодо заяви". Отримана на цьому етапі функція "2.2.5. Дозволити ремонт" є атомарною функцією. Ремонт дозволяється або не дозволяється.

При розгляді ієрархії функцій розробникові БД варто звернути увагу на наступні моменти:

- у функціональній моделі БД описуються бізнес-функції, і не всі вони будуть безпосередньо підтримуватися додатком БД;
- при розгляді ієрархій нерідко виникає ситуація, коли екземпляри однієї й тієї ж функції будуть мати різні номери.

Якщо в першому випадку додаткову інформацію про те, які бізнес-функції будуть реалізовані в системі, можна одержати від керівника проекту, то в другому випадку розробник БД, найімовірніше, має справу з помилкою аналітика у визначенні функції.

4.2 Визначення функцій

При розробці ієрархії функцій аналітик повинен надати текстовий опис до кожної функції, принаймні для

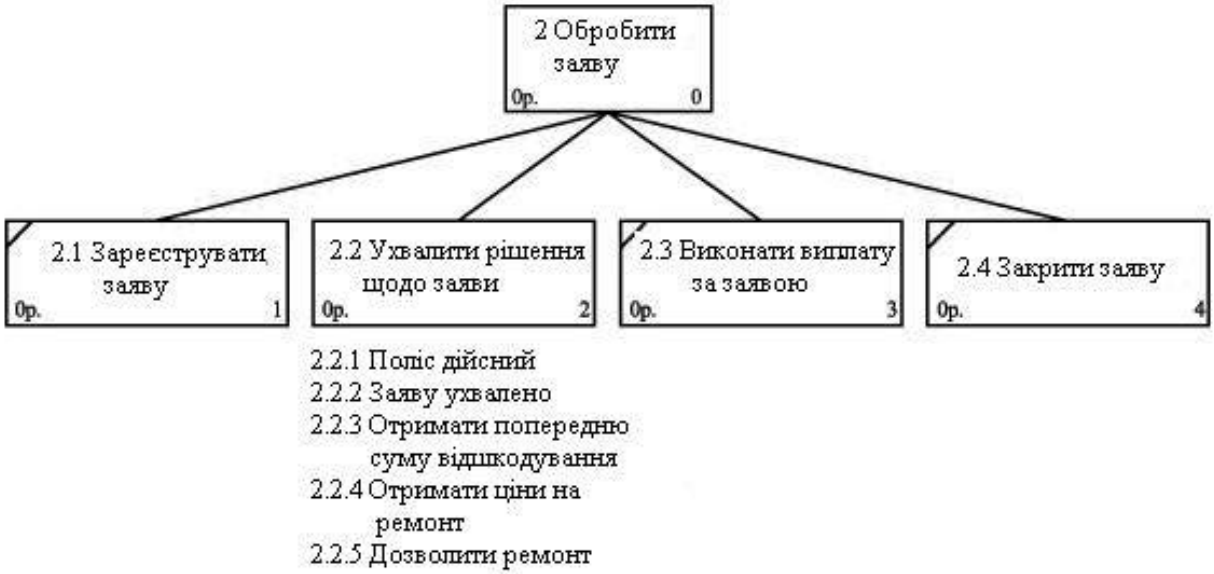


Рис. 4.1. Ієрархія функції для обробки заяв про виплату страхового відшкодування

верхнього й самого нижнього рівнів ієрархії. Бажано, щоб у цьому описі аналітики виділяли сутності ПО. Це важливо для того, щоб знати з якими сутностями ПО працює функція, тобто які потенційні об'єкти реляційної БД будуть використовуватися в кожній функції. Якщо це не зроблено, то розробникам БД прийде робити це самостійно.

Приклад. *Визначення функції "2.2.2. Перевірити чи ухвалена заява".*

"Одержати й зареєструвати всі необхідні страховою компанією відомості про заяву (ВІДОМОСТІ ПРО ЗАЯВУ), включаючи всі докладні відомості про треті сторони (СТОРОННІ ЮРИДИЧНІ ОСОБИ) і свідках (ФІЗИЧНІ ОСОБИ).

Вивчити страховий поліс (ПОЛІС) на предмет наявності виняткових ситуацій (ВИКЛЮЧЕННЯ) і визначити, чи діють ці ситуації у випадку даної заяви (ЗАЯВА).

Якщо є виключення, то закрити заяву й скласти стандартний лист заявникові про відмову у виплаті (ЛИСТ) заявникові (ЗАЯВНИК).

Якщо ніяких виключень ні, то змінити статус заяви на очікування оцінки, призначити й повідомити оцінювача (ОЦІНЮВАЧ)."

Із приклада видно, які сутності ПО беруть участь у виконанні функції (виділені в дужках), як міняється стан сутності (виділено курсивом) і який алгоритм роботи цієї функції.

Із приклада зрозуміло, що на цьому етапі розробник БД у якості вхідних даних використає також інформаційну модель ПО БД (опис сутностей).

При виконанні аналізу функцій корисно мати деяку таблицю (матрицю) "Функція-Сутність". Ця матриця повинна дати відповідь на наступні питання:

- чи має кожна сутність конструктор (функцію, що створює всі екземпляри сутності);
- чи має вона деструктор (функцію, що видаляє екземпляри сутності);
- є чи посилання на цю сутність (функції, які використають цю сутність й який образ).

Процес аналізу взаємодії функції й сутності прийнята позначати абревіатурою CRUD (Create, Reference, Update, Delete - створення, посилання, модифікація, видалення).

Корисними для розуміння розробником бази дані призначення функцій і того, як дані функції беруть участь у процесі обробки даних у системі, можуть бути діаграми потоку даних і діаграми життєвих циклів сутностей, які були розглянуті нами в другій лекції. Останні, зокрема, дають ясну картину зміни стану сутності, що важливо у визначенні атрибутів статусу сутності.

4.3 Відображення функцій у модулі

Одним з основних завдань проектування модулів додатків є побудова відображення функцій у модулі. При вирішенні цього завдання розробник БД повинен акцентувати увагу на структурі БД, що становить основу додатка.

Як правило, вирішення завдання відображення функцій у модулі виконується в чотири етапи:

1. Аналіз роботи функції.
2. Побудова моделі сутностей, що підтримує ці функції.
3. Почати проектування фізичної структури зі створення схеми, що підтримує розроблену модель сутностей.

4. Завершити проектування розробкою специфікацій модулів, які реалізують функції на запропонованій схемі БД.

Із запропонованого вище підходу видно, як тісно переплітаються у процесі проектування процеси розробки фізичної моделі БД і специфікацій модулів додатків. Таким чином, якщо розробником був розроблений чорновий варіант фізичної моделі БД по алгоритмах, розглянутим нами в попередніх лекціях, то на цьому етапі він повинен бути адаптований до реалізації функцій й, можливо, значно перероблений.

При відображенні функцій у модулі необхідно отримати схему, що ставить у відповідність кожної функції певний модуль.

Розглянемо нашу БД, що містить інформацію про співробітників, відділи й проекти організації. Припустимо, вона буде підтримувати бізнес-функцію "Керування проектами в організації". Функціональна модель ПО БД у термінах ієрархії функцій наведена на рис. 4.2, а на рис. 4.3 наведений перелік функцій керування проектами в організації.

Завдання полягає у відображенні функцій з переліку на рис. 4.3 у перелік модулів. Спочатку з переліку функцій повинні бути вилучені ті функції, які не будуть підтримуватися додатком БД. Розробник довідається в керівника проекту, що в додатку БД не будуть підтримуватися наступні функції:

- призначити куратора проекту; сповістити керівників підрозділів;
- сповістити співробітників; зібрати нарада;
- приступитися до виконання; скласти список робіт;
- визначити обсяг робіт; визначити вартість робіт;

Рисунок 4.2 - Ієрархія бізнес-функції "Управління проектами в організації"

Рисунок 4.3 - Перелік функції управління проектами в організації

- визначити час робіт; визначити виробничі потужності;

- розподілити виробничі потужності; розподілити роботи між співробітниками;
- контролювати хід виконання проекту.

Таким чином, буде отриманий перелік функцій, що показаний у лівій колонці таблиці 4.1. Цьому переліку функцій повинен бути поставлений у відповідність перелік модулів додатка БД.

Керівник проекту передав розробникові БД характеристику додатка БД щодо управління виконанням проектів в організації. У цьому додатку буде здійснено облік виконуваних і виконаних проектів в організації.

Розробник БД повинен встановити відображення функцій у модулі, як показано на рис. 4.4.

Наведений приклад показує загальний принцип побудови відображення бізнес-функцій у модулі.

Таблиця 4.1 – Переліки функцій і модулів

Функції	Модуль
Призначити керівника проекту	Введення інформації про проект
Визначити бюджет проекту	Введення інформації про співробітників
Визначити список підрозділів	Пошук інформації про співробітників
Визначити список співробітників	Пошук інформації про проекти
Виконувати проект	Генерація звіту про виконані проекти
Здати проект	Генерація звіту про виконувані проекти

Рисунок 4.4 – Відображення функції в модулі

Список використаної літератури

1. К. Дж. Кейт. Введення в системи баз даних/ пер. с англ.− 8-е изд. − М.: Издательский дом «Вильямс», 2006.– 1328 С.
2. Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: учебное пособие/ Изд-е Башкирского ун-та. - Уфа, 1999. - 108 с.
3. Пушников А.Ю. Введение в системы управления базами данных. Часть 2. Нормальные формы отношений и транзакции: учебное пособие/ Изд-е Башкирского ун-та. - Уфа, 1999. - 138 с.
4. Мартин Грубер. Понимание SQL. /пер. В.Н. Лебедева.− М., 1993.– 291 с.
5. Томас Коннолли, Каролин Бегг. Базы данных. Проектирование, реализация и сопровождение. Теория и практика.– 3-е изд.− М.: Издательский дом «Вильямс», 2003. − 1436 с.
6. Джен Л. Харрингтон. Проектирование реляционных баз даннях. − М.: Издательство «Лори», 2006. − 230 с.
7. Киммел Пол. Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа.: пер. с англ.- М.: Издательский дом «Вильямс», 2003.- 480 с.: илл.- Парал. тит. англ.
8. Кириллов В.В. Основы проектирования реляционных баз даннях: учебное пособие. - СПб.: ИТМО, 1994. – 90 с.
9. В.В. Кириллов, Г.Ю. Громов. Учебное пособие по SQL: Структурированный язык запросов (SQL). http://www.citforum.ru/database/sql_kg/index.shtml

10. Пасічник В.В.Організація баз даних та знань: підручник для ВНЗ/ В.В. Пасічник, В.А. Резніченко.-К.: Видавнича група BHV,2006.-384с.

11. Бекаревич Ю.Б., Пушкина Н.В. Microsoft Access 2000.- СПб.: БХВ – Санкт- Петербург, 1999.- 480 с., ил.

Навчальне видання

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Конспект лекцій

для студентів спеціальності
050101 – Інформаційні технології проектування
заочної форми навчання

Відповідальний за випуск І.В. Баранова
Редактор Н.В.Лисогуб
Комп'ютерне верстання А.В. Нені

Підписано до друку 20.04.2010, поз.
Формат 60х84/16. Ум. друк. арк. __ Обл.-вид.арк.___ Тираж 50 пр.
Зам.№
Собівартість вид.___ грн___ к.

Видавець і виготовлючач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК №3062 від 17.12.2007.